On P Systems with Bounded Parallelism

Francesco Bernardini*, Francisco J. Romero-Campero†, Marian Gheorghe*,
Mario J. Pérez-Jiménez†, Maurice Margenstern‡, Sergey Verlan‡,
Natalio Krasnogor§
*Department of Computer Science, The University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
Email: {F.Bernardini, M.Gheorghe}@dcs.shef.ac.uk
†Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville, Avda. Reina Mercedes, 41012 Sevilla, Spain
Email: {fran, marper}@cs.us.es
‡ LITA, Université de Metz
Ille du Saulcy, 57045 Metz Cedex, France
E-mail: {margens, verlan}@sciences.univ-metz.fr
§ Automated Scheduling, Optimisation and Planning Research Group
University of Nottingham, Jubilee Campus, Nottingham NG8 1BB, UK
E-mail: Natalio.Krasnogor@nottingham.ac.uk

***Abstract*— A framework that describes the evolution of P systems with bounded parallelism is defined by introducing basic formal features that can be then integrated into a structural operational semantics. This approach investigates a generic strategy of selecting membranes and rules and applying the rules. P systems with boundary rules are used to illustrate the case and an example dealing with an evolution strategy involving bounded parallelism is discussed.**

I. Introduction

Membrane computing has been introduced with the aim of defining a computing device, called P system, which abstracts from the structure and the functioning of living cells [21]. The main results in this area show that P systems are a very powerful computational model mostly equivalent to Turing machines. Recent researches have been instead dedicated to the study of P systems as a modelling tool for biological systems [7], [8], [24], [5], [9]. In this case P systems are not used as a computing paradigm, but rather as a formalism for describing the behaviour of the system to be modelled. Therefore, there is a growing interest in developing implementations for the membrane computing paradigm in order to be able to execute a P system model and run simulations of biological phenomena. In this respect, a number of tools have already been produced (some of them are available from [29]) but yet correct implementation techniques need to be identified, especially when the quantitative aspects featuring the "reality" of a biological phenomenon are considered by the model.

Apart from the main stream of the research involving the study of the computational capabilities of different variants of P systems and applications in biology, there were investigations into formally defining the semantics of the behaviour and evolution of such systems. In [2] an operational semantics for a class of P systems is defined and its implementation in Maude is described. A step further has been taken by providing a structural operational semantics for that class of P systems with correctness proofs for each set of inference rules [3]. Both approches refer to P systems evolving in a maximally parallel manner, which is the cornerstone of most of the variants of the model.

In this paper, we present a variant of P systems with boundary rules where the rules have associated conditions and attributes. Then, for this variant, we provide some operational semantics definitions that characterise different strategies for the application of the rules based on the notion of parallelism of type (k, q) (in each step, at most k membranes can evolve and, inside each one of them, at most q rules can be used). The selection of membranes and rules relies on the structure of the systems and conditions and attributes associated to rules. This evolution strategy will cover some more recent approaches dropping the maximal parallelism and considering asynchronous sequential behaviour [13], bounded parallelism [12], probabilistic approaches [25], biologically oriented methods [8], [10], [5], [24].

II. The Model

A cell-like P system is usually defined as a hierarchical arrangement of a number of membranes identifying a corresponding number of regions inside the system, and with these regions having associated a finite multiset of objects and a finite set of rules. Moreover, in such a *membrane structure*, all the membranes but one must be included in a unique main membrane, which defines the "boundary" of the system with respect to the external environment. This particular membrane is called *skin membrane*. Rules of many different forms have been considered for cell-like P systems in order to encode the operation of modifying the objects inside the membrane,

moving objects from one place to the other, dissolving, creating, dividing membranes etc. The former operation is usually encoded by using multiset rewriting rules whereas, for the movement of objects across the membranes, one can either choose to use the targets $here, in, out$, which represent a simple model for the biological process of diffusion of molecules across the membranes, or to use symport/antiport rules, which provide an abstraction for the biological mechanisms of active transport of small molecules across the membranes [22].

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

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

with u, v, u', v' some finite multisets of objects and l the label of a membrane. These rules are multiset rewriting rules that operate on both sides of the membranes, that is, a multiset u placed outside a membrane labelled by l and a multiset v placed inside the same membrane can be simultaneously replaced by a multiset u' and a multiset v' respectively. In this way, we are able to capture in a concise way the features of both the communication rules and the transformation rules considered in [6]. Moreover, as shown in [24], rules like (1) allow us to express any sort of interactions occurring at the membrane level, and, in particular, they are useful to model the binding of a signal molecule to its corresponding receptor that occurs at the cell-surface level.

Another important concept commonly used in membrane computing is that of promoters/inhibitors as multisets of objects that affect the applicability of the rules associated to membranes. Promoters are multisets of objects which, when present inside a membrane, make certain rules to be applicable inside that membrane and which prevent those rules from being applied when they are not present inside that membrane. Inhibitors are instead multisets that operate in the opposite sense: when they are present inside a membrane they block the application of certain rules inside that membrane [22]. We generalise this concept of promoters/inhibitors by associating to each rule a boolean predicate π expressing a generic property over the objects contained inside a membrane and the objects contained in the surrounding region or in one of the regions that exists inside the current one. Such a predicate π is meant to specify a condition that need to be satisfied to make the rule to be applicable inside a given membrane.

Finally, we associate to each rule a finite set of attributes which are meant to capture the quantitative aspects that are often necessary to characterise the "reality" of the phenomenon to be modelled. The necessity of taking into account these quantitative aspects has been made clear in a few recent application of P systems to the modelling of biological systems [7], [8], [24], [5], [9].

Therefore, we introduce the following notion of program as the basic feature describing a generic process occurring inside a membrane.

Definition 2.1 (program): Let O be an alphabet for the objects and let Lab be an alphabet of labels. A program is a construct

$$\langle \pi \;>>\; u\,[_l\,v\,] \rightarrow u'\,[_l\,v'\,], A \rangle$$

with $u, v, u', v' \in O^*$ some finite multisets of objects, π a generic boolean predicate, l is a label from Lab and A a finite set of attributes associated with the rule.

As we have already said, the predicate π is used to express a condition that needs to be satisfied in order to make the rule applicable inside a membrane. A rule contained in a program is said to be active inside a membrane if the predicate π evaluates to true, otherwise it is said to be inactive. The set of attributes in A can instead be used to associate to each rule a kinetic constant [8], [24], [5] a probability [7], [4], or a more general function returning the number of occurrences of the multisets u, v to be consumed and the number of occurrences of the multisets u', v' to be produced. As well as this, the attributes might be used to associate to a rule some "side-effect" in order to alter other properties of the membrane where the rule is applied.

Our notion of program bears some similarities with, and is somehow inspired by, the notion of attribute grammars used for syntax-directed language translation and automatic code generation [1], as well as the notion of parametric L systems augmented with C code used for modelling plant growth and development [17]. In a sense, our programs also resemble the concept of guarded commands for non-deterministic programming introduced by Dijkstra in 1975 which has then led to CCS, CSP and the modern theory of concurrent systems based on π-calculus [20].

Now, we can define a P system by simply associating a finite multiset of objects to each membrane in a given membrane structure and by considering a finite set of programs to make these objects evolve from one configuration to the other.

Definition 2.2 (P system): A P system is a construct $\Pi = (O, Lab, \mu, M_1, M_2, \ldots, M_n, R_1, \ldots, R_n)$ where:

- O is a finite alphabet of symbols representing objects;
- Lab is a finite alphabet of symbols representing labels for the compartments;
- μ is a membrane structure containing $n \geq 1$ membranes labelled with elements from Lab;
- $M_i = (w_i, L_i)$, for each $1 \leq i \leq n$, is the initial configuration of membrane i with $L_i \in Lab$ and $w_i \in O^*$ a finite multiset of objects;
- R_i, for each $1 \leq i \leq n$, is a finite set of programs in membrane i of the form specified in Definition 2.1 with objects in O and labels in Lab.

Thus, the initial configuration of each membrane i, with $1 \leq i \leq n$, is given by a finite multiset of objects from O and by a label from Lab. A program consists of a rule of the form $u\,[_{L_i}\,v\,] \rightarrow u'\,[_{L_i}\,v'\,]$, with L_i the label of membrane i, may be either in L_i or in the membrane surrounding it. Moreover, we can now precisely say that the evaluation of the predicate π in such a program must be done by considering both the content of membrane i and the content of the membrane $upper(\mu, i)$ (i.e., the membrane that directly contains membrane i). The evaluation of the predicate π, which has to be done before the application of the rule inside membrane i, is then denoted by $\pi(i, upper(\mu, i))$.

P systems are usually considered as being distributed maximal parallel multiset rewriting systems [22]. This means, in each step, in each membrane, all the objects that can evolve by

means of some rules must evolve in parallel at the same time by means of these rules, with the sole restriction that the same occurrence of the same object cannot be used by more than one rule at a time. That is, in each step, for each membrane, a maximal set of rules to be applied is non-deterministically selected by making sure that no further rules can be applied to the objects left inside the membranes. On the other hand, a few recent works [18], [7], [8], [25], [24], [5] have addressed the issue of introducing new strategies for the application of the rules where the set of rules to be applied in any step is not maximal, but it is somehow bounded. The reasons for the introduction of new derivation/evolution strategies may be different, but, in the context of modelling biological systems, one can say that restrictions to maximal parallelism are often required in order to close the gap between the abstractness of the model and the "reality" of the phenomenon to be modelled [18], [7], [8], [25], [24], [5]. In this respect, the key issues that need to be addressed in order to define a strategy for the application of the rules in a P system are: a) how to select the next rule to be applied inside a given membrane, b) how many different rules can be applied in parallel at the same time inside the membrane, and c) how many different membranes can evolve in parallel at the same time.

A function f will compute the number of membranes that will be selected to evolve in each step of the computation (k) and the number of rules applied in each membrane (q). For this reason this concept of parallelism will be called of type (k, q). This function may return a bounded number of rules that are applied a certain number of times or an arbitrary number of times or in a maximal parallel manner. The rules may be chosen non-deterministically or according to some restrictions imposed (partial order or priority relationships) or some other criteria that might rely on attribute values leading to different selection mechanisms.

The function f is given in every case by a precise algorithm. An illustration will be further given in the paper.

These strategies are presented in the framework of an operational semantics very similar to the one proposed in [2]. The approach will provide a basis to inference rules defined in [3].

III. Operational Semantics

We will introduce some concepts that define the framework of an operational semantics for P systems evolving in a $(k, q)-$parallel manner.

Let $\Pi = (O, Lab, \mu, M_1, M_2, \ldots, M_n, R_1, \ldots, R_n)$ be a P system as specified in Definition 2.2, where $M_i = (w_i, L_i)$ with $1 \leq i \leq n$ and the programs in R_i are denoted

$$j : \langle \pi >> u\,[_{L_i}\, v\,] \rightarrow u'\,[_{L_i}\, v'\,], A\rangle.$$

Similar to [3] we will denote by O_c^* the set of finite multisets over O. If a membrane M_i contains the multiset wuw' and the membrane M_l, included in M_i, has the multiset yvy' and R_i contains $j : \langle \pi >> u\,[_{L_l}\, v\,] \rightarrow u'\,[_{L_l}\, v'\,], A\rangle$ and the predicate π is true then the result of applying j to objects contained in M_i and M_l simultaneously will lead to the multisets $w(u', here)w'$ and $y(v', here)y'$ in M_i and M_l, respectively. I.e. we will attach to any multiset obtained by applying a given program the target $here$. This will allow when the structural operational semantics is defined according to [3] to simulate the execution of a parallel rewriting rule in two steps: **rewriting** and **communication**. After the first step the multisets obtained will have associated the target $here$ and in the second step these will be reverted to usual multisets over O [3].

It will be also defined, similar to [3], the concept of committed configuaration and the process of passing from one configuration C_1 to another one C_2, $C_1 \Longrightarrow C_2$, if the next two steps are executed

- $(k, q)-$parallel rewriting step, $C_1 \overset{pr}{\Longrightarrow}_{(k,q)} C_1'$, consists of selecting k membranes and in each membrane q rules are applied according to the function f;
- parallel communication of objects, $C_1' \overset{tar}{\Longrightarrow} C_2$, consists of reverting the objects marked as $here$ to objects of O.

The concepts of **irreducibility** and $(L, w)-$**consistency** will be introduced similar to [3], where L is the label of a membrane and w its current multiset of objects. The concept of **mpr-irreducibility** will be replaced by $(k, q)-$**irreducibility**. In this case a multiset consisting only of objects is called $L-$**irreducible** iff q rules have been applied or less than q when there are no more rules available. For a membrane M labelled L and containing the multiset w, a non-empty multiset of n boundary rules is $(L, w)-$**consistent** iff the left hand side of these rules have the multisets u_i, their predicates are true and:

- $w = u_1 \ldots u_n z$, so each rule is applicable to w,
- either $n = q$ or $n < q$ and no other rules are available to be applied.

For example the P systems $\Pi = (O, Lab, \mu, M_1, M_2, \ldots, M_n, R_1, \ldots, R_n)$ where

- $O = \{a, b, c, d, e, g, x, y\}$;
- $Lab = \{1, 2, 3\}$;
- $\mu = [_1[_2][_3]]$;
- $w_1 = a^5b^5, w_2 = c^4d, w_3 = eg$;
- $R_1 = \{a[_2c] \rightarrow x[_2y]\}$, $R_2 = \{b[_2c] \rightarrow y[_2x]\}$, $R_3 = \emptyset$.

We will obtain for $(2, 2)-$parallel rewriting:

$$(1|a^5b^5; (2|c^4d); (3|eg)) \overset{pr}{\Longrightarrow}_{(2,2)}$$

$$(1|aaa(xx, here)bbb(yy, here)(2|(xxyy, here)d); (3|eg)).$$

Where the two membranes selected were 1 and 2 and in each of these membranes two rules are applied in parallel.

Having these concepts we may rewrite the inference rules [3] by replacing maximal parallelism with $(k, q)-$parallelism. It may be also proved that $(k, q)-$irreducibility induces correct operations. In the same way parallel communication may be approached (in this case only $here$ targets being considered) and correctness results for target irreducibility may be obtained similar to [3].

Remark 3.1: The approach on semantics proposed here for parallelism of type (k, q) includes the usual notion of maximal parallelism [22] as a particular case of it. In fact, it is easy to see that, when k is equal to the number of membranes in the systems and f returns the number of objects

in each membrane, our semantics is equivalent to the usual notion of maximal parallelism. In general, for such a type of parallelism, a rule contained in a program labelled by j is non-deterministically applied a number of times. It can be also considered a restriction on the application of a rule to a limited number of times.

Remark 3.2: The semantics approach proposed here, with respect to [2], defines a more general framework which abstracts from the language chosen for implementing the P system model; as well as this, the rules considered are of a more general form, which allow the simultaneous rewriting of objects on both sides of the membranes.

Remark 3.3: Our notion of parallelism of type (k, q) is somehow related to the notion of asynchronous P systems from [22] (see also [13] for some further discussions and results). The difference is that, for parallelism of type (k, q), the maximal number of membranes that can evolve in parallel and the maximal number of rules that can be applied inside these membranes are provided through the function f, whereas, for asynchronous P systems, the values of these constants are totally arbitrary; in fact, in each step, the number of membranes, as well as the number of rules to be applied inside these membranes are arbitrary. More recently, a notion of minimal parallelism has been proposed in [11] that restricts this latter type of non-determinism to the number of rules to be applied inside the membranes: in each step, all the membranes that can evolve by means of some rules must evolve but, inside each membrane, the number of rules to be applied is non-deterministically chosen. Finally, it is easy to see that the notion of P systems working in the sequential mode from [22], [13], [12] corresponds to the semantics proposed here for P systems operating with parallelism of type $(1, 1)$. Moreover, it is easy to see that our notion of parallelism of type (k, q) coincides with the notion of q-**Max-Parallelism** from [16] when k is equal to the number of membranes in the system.

Remark 3.4: An application of the semantics of P systems defined in this paper is considered in [24] where a model for the EGFR signalling cascade is proposed. In that case, rules are supposed to model chemical reactions with an associated kinetic constant and, for each rule, the function f returns all the membrane components and computes the so called "mass action law": the value is given by the product of the concentration of the reactants, expressed in molar units, multiplied by the corresponding kinetic constant. A slightly different approach is considered in [8] (see also [10], [9]) that is based on the notion of metabolic algorithm. In this case, all the rules are supposed to be applied in parallel at the same time with a multiplicity again given according to the mass action law. The variation on the concentration of each reactant is then obtained by summing up the contribution of each rule. In this way, it is avoided the introduction of the non-determinism, but it is necessary to introduce some further constraints in order to avoid consuming more reactants than those currently available inside the membranes.

IV. P SYSTEMS WITH PARALLELISM OF TYPE $(k, 1)$

In this section we will focus on a specific behaviour of P systems when in a number of membranes, most frequently only one, a program will be chosen to be executed. This example will illustrate a particular way to compute the function f that selects membranes and programs and it is fully chemically motivated.

A. An algorithm to select membranes and programs

Gillespie's algorithm [15] (see also [14] for some recent improvements) provides an exact method for the stochastic simulation of systems of bio-chemical reactions; the validity of the method is rigorously proved and it has been already successfully used to simulate various biochemical processes [19]. As well as this, Gillespie's algorithm is used in the implementation of stochastic π-calculus [26], [30] and in its application to the modelling of biological systems [27]. Our method based on Gillespie's algorithm represents a first attempt to simulate the behaviour of P systems with a stochastic semantics. This is done by taking into account the fact that, with respect to the original algorithm, in P systems we have different regions, each one with its own set of rules, and the application of a rule inside a region can affect the content of another region too. Specifically, let $\Pi = (O, Lab, \mu, M_1, M_2, \ldots, M_n, R_1, \ldots, R_n)$ be a P system as specified in Definition 2.2 with the membranes $M_i = (w_i, L_i)$ and the programs R_i, $1 \leq i \leq n$. The set R_i of programs that are active inside membrane contains elements of the form $(j, \pi_j, r_j, p_j, k_j)$ where:

- j is the index of a program from R_i;
- π_j is the predicate; in this section this will be always true and will be omitted;
- r_j is the boundary rule rule contained in the program j;
- p_j is the probability of the rule contained in the program j to be applied in the next step of evolution; this probability is computed by multiplying a stochastic constant k_j, specifically associated with program j, by the number of possible combinations of the objects present on the left-side of the rules with respect to the multiset w_i (or the multiset $w_{i'}$, with $i' = upper(\mu, i)$) - the current content of membrane i (i'); for example, if we have a rule $[\, ab\,]_{L_i} \rightarrow [\, v\,]_{L_i}$, with $a, b \in O$, $v \in O^*$, the probability p_j is given by $k_j * |w_i|_a * |w_i|_b$ (i.e., there are $|w_i|_a * |w_i|_b$ different possible ways of assigning objects to the rule $[\, ab\,]_{L_i} \rightarrow [\, v\,]_{L_i}$).

Then an algorithm to select membranes and programs for $(k, 1)-$parallel behaviour of a P system is introduced.

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

1) calculate $a_0 = \sum p_j$, for all $(j, r_j, p_j, k_j) \in R_i$;
2) generate two random numbers r_1 and r_2 uniformly distributed over the unit interval $(0, 1)$;
3) calculate the waiting time for the next reaction as $\tau_i = \frac{1}{a_0} ln(\frac{1}{r_1})$
4) take the index j, of the program such that $\sum_{k=1}^{j-1} p_k < r_2 a_0 \leq \sum_{k=1}^{j} p_k$;

5) return the triple (τ_i, j, i).

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

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

- **Initialisation**
  - set time of the simulation $t = 0$;
  - for each membrane i in μ compute a triple (τ_i, j, i) by using the procedure described above; construct a list containing all such triples;
  - sort the list of triple (τ_i, j, i) according to τ_i;
- **Iteration**
  - extract the first triple, (τ_m, j, m) from the list;
  - set time of the simulation $t = t + \tau_m$;
  - update the waiting time for the rest of the triples in the list by subtracting τ_m;
  - apply the rule contained in the program j only once changing the number of objects in the membranes affected by the application of the rule;
  - for each membrane m' affected by the application of the rule remove the corresponding triple $(\tau'_{m'}, j', m')$ from the list;
  - for each membrane m' affected by the application of the rule j re-run the Gillespie algorithm for the new context in m' to obtain $(\tau''_{m'}, j'', m')$, the next program j'', to be used inside membrane m' and its waiting time $\tau''_{m'}$;
  - add the new triples $(\tau''_{m'}, r'', m')$ in the list and sort this list according to each waiting time and iterate the process.
- **Termination**
  - Terminate simulation when time of the simulation t reaches or exceeds a preset maximal time of simulation.

Therefore, in this approach, it is the waiting time computed by the Gillespie's algorithm to be used to select the membranes which are allowed to evolve in the next step of computation. These membranes have the same waiting time. Specifically, in each step, the membranes associated to programs with the same minimal waiting time are selected to evolve by means of the corresponding rules. Moreover, since the application of a rule can affect more than one membrane at the same time (e.g., some objects may be moved from one place to another), we need to reconsider a new program for each one of these membranes by taking into account the new distribution of objects inside them.

Remark 4.1: The use of a variable time-unit for each step does not affect the semantics of our model; in each step, a single rule at a time is applied inside a specific membrane. This means the behaviour of the systems is still synchronous although each application of the rule has associated a different time-unit. In fact, the waiting time is mainly used as a parameter necessary to determine the rule to be applied in the next step of computation.

Remark 4.2: The current algorithms brings some improvements with respect to the general strategy of selecting membranes and programs. In the iteration phase, we need not to recompute all the probabilities associated with each program applicable inside each membrane, but we can do that only for those membranes which are actually affected by the last application of a program. That is because the value of the probabilities associated with the other rules remain unchanged.

Remark 4.3: The use of the waiting time parameters leads to selecting membranes using the minimum waiting time principle in order to chose those with the same value of that parameter. Getting rid of this parameter will lead to a variant of this algorithm that is associated to a $(n, 1)-$parallel behaviour of the system, where n is the total number of membranes. Indeed, in this case there is no way to distinguish between membranes and all them will be selected.

B. Example

A simulator for P systems that is based on the semantics illustrated has been implemented in Scilab [28]. The current simulator has been implemented only for P systems with programs where all the rules have one of the following forms:

$$[\,v\,]_l \to [\,v'\,]_l,\ u\,[\,]_l \to [\,v'\,]_l,\ [\,v\,]_l \to u'\,[\,v'\,]_l$$

that is, rules that do not allow the simultaneous rewriting of objects both inside and outside a membrane. As well as this, all the programs are supposed to be without conditions. We aim at producing a full implementation of programs specified as in Definition 2.1 in a later version of the simulator, which is now under development.

A colony of *vibrio fischeri* bacteria is modelled in [5] by considering a number of elementary membranes, each one of them representing a bacterium, included in an unique main membrane, which represents the environment. Each elementary membrane contains a model of the gene regulatory network which controls the quorum sensing process in such a bacterium. Here we do not illustrates the details of this model from [5] but we just report the results of the simulations.

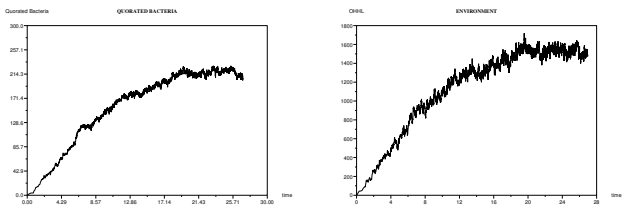

Fig. 1. **Gillespie-like**: The increase on the number of quorated bacteria in correspondence of the increase on the number of signal molecules in the environment.

Figure 1 reports the simulation results for a colony of 300 bacteria obtained by using Gillespie's algorithm implementation. This shows how the implementation is able to reproduce the expected behaviour: the quorum sensing process takes place through a progressive increase on the number of signal

molecules in the environment that leads to an increase on the number of quorated bacteria.

V. Conclusions

We have introduced a class of P systems with bounded parallelism ans a framework for an operational semantics which captures different strategies of applying the rules based on the notion of parallelism of type (k, q). This is particularly important in the context of P system developments as it shows another way of evolving such systems opposed to the maximal parallelism defined formally in a number of approaches dealing with semantics [2], [3]. It also unifies a number of attempts to define variants of P systems with bounded parallelism, or relying on probabilistic behaviour. On the other hand refers to a specific class of functions selecting membranes and programs to be executed, that are fully chemically and biologically motivated.

In terms of applications, our approach points out some key-issues which need to be addressed every time a new model for a biological system is developed:

- How do we represent the "reality" of the phenomenon in a P system model? Which variant of P systems must we consider?
- What strategy for the application of the rules do we intend to adopt? What are the alternatives? How do we implement them? How do we evaluate their performances?
- If a particular simulation does not reproduce the "expected" behaviour, what do we need to change? Do we need to change the model or do we need to try a different strategy for the application of the rules? How do the choice of parameters affect the result of a simulation?

These questions can be answered only through empirical research which attempts to match the results of a P system simulation with the results of biological experiments and/or the evidences resulting from other well-consolidated mathematical models. On the other hand, more ambitious researches should be directed to the development of appropriate "formal" validation/verification techniques, which possibly would allow us to formulate reasonable hypotheses with respect to the behaviour of biological systems. Attempts to use model checking in this context have been initiated by [2].

Acknowledgement. The authors wish to thank the anonymous referees for their valuable comments.

References

[1] Aho, A.V., Sethi, R., Ulmann, J.D. (1986). *Compilers: Principles, Techniques, and Tools*. Addison-Wesley.

[2] Andrei, O., Ciobanu, G., Lucanu, D. (2004). Executable Specifications of P Systems. In [18], 126–145.

[3] Andrei, O., Ciobanu, G., Lucanu, D. (2005). Structural Operational Semantics of P Systems. In: *Pre-Proceedings of WMC6 - Vienna 2005*, 1–23 .

[4] Ardelean, I., Cavaliere, M. (2003). Playing with a Probabilistic P Simulator: Mathematical and Biological Problems. In: *Brainstorming Week on Membrane Computing, Tarragona, Feb 5-11 2003* (Cavaliere, M., Martin-Vide, C., Păun, Gh., eds.). Tech. Rep. **26/03**, Universitat Rovira i Virgili, Tarragona, Spain, 37–45.

[5] Bernardini, F., Gheorghe, M., Muniyandi, R.C., Krasnogor, N., Pérez-Jiménez, M.J., Romero-Campero, F.J. (2005). On P Systems as a Modelling Tool for Biological Systems. In: *Pre-Proceedings of WMC6 - Vienna 2005*, 193–213.

[6] Bernardini, F., Manca, V. (2003). P Systems with Boundary Rules. In: [23], 107–118.

[7] Besozzi, D. (2004). *Computational and Modelling Power of P systems*, Ph.D. Thesis, Università degli Studi di Milano, Milan, Italy.

[8] Bianco, L., Fontana, F., Franco, G., Manca, V. (2005). P Systems for Biological Dynamics. In: *Applications of Membrane Computing* (Ciobanu, G., Păun, Gh., Pérez-Jiménez, M.J., eds.), Springer-Verlag, Berlin, Heidelberg, New York, 81–126.

[9] Bianco, L., Fontana, F., Manca, V. (2005). P Systems and the Modelling of Biochemical Oscillation. In: *Pre-Proceedings of WMC6 - Vienna 2005*, 214–225.

[10] Bianco, L., Fontana, F., Manca, V. (2005). Metabolic Algorithm with Time-varying Maps. In: *Proceedings of the Third Brainstorming Week on Membrane Computing. Sevilla (Spain), January 31 - February 4, 2005.* (Gutiérrez-Naranjo, M.A., Riscos-Núñez, A., Romero-Campero, F.J., Sburlan, D.,eds.), University of Seville, Seville, Spain, 43–62.

[11] Ciobanu, G., Păun, Gh. (2005). The Minimal Parallelism is Still Universal. Submitted.

[12] Dang, Z., Ibarra, O. (2005). On P Systems Operating in Sequential-Mode. *International Journal of Foundations of Computer Science*, to appear.

[13] Freund, R. (2005). Asynchronous P Systems and P Systems Working in the Sequential Mode. In: [18], 36–62.

[14] Gibson, M.A., Bruck, J., (2000). Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels. *Journal of Physical Chemistry*, **104**, 25, 1876–1889.

[15] Gillespie, D.T. (1977). Exact Stochastic Simulation of Coupled Chemical Reactions. *The Journal of Physical Chemistry*, **81**, 25, 2340–2361.

[16] Ibarra, O.H., Yen H.C., Dang, Z. (2005). On Various Notions of Parallelism in P Systems. *International Journal of Foundations of Computer Science*, **16**, 4, 693–705.

[17] Karwowski, R., Prusinkiewicz, P. (2003). Design and Implementation of the L+C Modelling Language. *Electronics Notes in Theoretical Computer Science*, **82**, 2, 1–19.

[18] Mauri, G., Păun, Gh., Pérez-Jiménez, M., J., Rozenberg, G., Salomaa, A., eds. (2005). Membrane Computing. International Workshop, WMC 2004, Milan, Italy, June 2004. Revised and Invited Papers. *Lecture Notes in Computer Science*, **3365**, Springer-Verlag, Berlin, Heidelberg, New York.

[19] Meng, T.C., Somani S., Dhar, P. (2004). Modelling and Simulation of Biological Systems with Stochasticity. *In Silico Biology*, **4**, 0024.

[20] Milner, R. (1999). *Communicating and Mobile System: The π-Calculus*. Cambridge University Press.

[21] Păun, Gh. (2000). Computing with Membranes. *Journal of Computer and System Sciences*, **61**, 1, 108–143.

[22] Păun, Gh. (2002). *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, Heidelberg, New York.

[23] Păun, Gh., Rozenberg, G., Salomaa, A., Zandron, C., eds. (2003). Membrane Computing. International Workshop, WMC-CdeA 02, Curtea de Arges, Romania, August 19-23, 2002. Revised Papers. *Lecture Notes in Computer Science*, **2597**, Springer-Verlag, Berlin, Heidelberg, New York.

[24] Pérez-Jiménez, M.J., Romero-Campero, F.J. (2005). Modelling EGFR Signalling Cascade Using Continuous Membrane Systems. In: *Proceedings of the Third International Workshop on Computational Methods in Systems Biology 2005 (CMSB 2005)* (Plotkin, G., ed.), University of Edinburgh, Edinburgh, United Kingdom.

[25] Pescini, D., Besozzi, D., Zandron, C., Mauri, G. (2005). Dynamical Probabilistic P Systems: Definitions and Application. In: *Proceedings of the Third Brainstorming Week on Membrane Computing. Sevilla (Spain), January 31 - February 4, 2005.* (Gutiérrez-Naranjo, M.A., Riscos-Núñez, A., Romero-Campero, F.J., Sburlan, D.,eds.), University of Seville, Seville, Spain, 275–287.

[26] Philips, A., Cardelli. L. (2004). A Correct Abstract Machine for the Stochastic Pi-calculus. *Electronical Notes in Theoretical Computer Science*, to appear.

[27] Priami, C., Regev, A., Shapiro, E., Silverman, W. (2001). Application of a Stochastic Name-Passing Calculus to Representation and Simulation of Molecular Processes. *Information Processing Letters*, **80**, 25–31.

[28] Scilab Web Pages: `http://scilabsoft.inria.fr`.

[29] The P Systems Web Page: `http://psystems.disco.unimib.it`.

[30] The Stochastic Pi-Machine: `http://www.doc.ic.ac.uk/~anp/spim/`.