Chapter 1
Kernel P Systems and Stochastic P Systems for Modelling and Formal Verification of Genetic Logic Gates

Marian Gheorghe, Savas Konur and Florentin Ipate

Abstract *P* systems are the computational models of *membrane computing*, a computing paradigm within natural computing area inspired by the structure and behaviour of the living cell. In this chapter, we discuss two variants of this model, a non-deterministic case, called *kernel P (kP)* systems, and a stochastic one, called *stochastic P (sP)* systems. For both we present specification laguages and associated tools, including simulation and verification components. The expressivity and analysis power of these natural computing models will be used to illustrate the behaviour of two genetic logic gates.

Key words: membrane computing, kernel P systems, stochastic P systems, model checking, specification / property language, genetic logic gates

1.1 Introduction

Membrane computing is a computational paradigm, within the more general area of natural computing [32], inspired by the structure and behaviour of eukaryotic cells. The formal models introduced in this context are called *membrane systems* or *P systems*. After their introduction [27], membrane systems have been widely investigated for computational properties and complexity aspects, but also as a model for various applications [28]. Many different variants of P systems have been introduced and studied, mainly due to the many theoretical challenges induced by them, but also motivated by the need to model different problems. Most of these variants of P systems consider key features of the biological cell as part of the computational models introduced. In this respect, they deal with either simple bio-chemical elements (called *objects*) or more complex molecules like DNA strands which are codified as *strings*. Specific molecules like catalysts, activators and inhibitors are also utilised by the

Marian Gheorghe, Savas Konur
School of Electrical Engineering and Computer Science, University of Bradford, Bradford BD7 1DP, UK
e-mail: {m.gheorghe, s.konur}@bradford.ac.uk

Florentin Ipate
Department of Computer Science, University of Bucharest, Str. Academiei nr. 14, 010014, Bucharest, Romania
e-mail: florentin.ipate@ifsoft.ro

models. Chemical interactions within compartments and transmembrane regulations are represented by *rewriting rules* and *comunication rules*, respectively. Various biological entities, like, cells, tissues, as well as specialised cells, such as neurons, are described in the membrane systems framework by *cell P systems*, *tissue P systems*, and *neural P systems*, respectively. The combination of these features leads to a rich set of variants of P systems. A thorough presentation of the theoretical developments is provided in [28], whereas various applications of this computing paradigm in modelling problems from various areas, including computer science, graphics and linguistics, can be found in [8]. More recently, it has been applied to systems and synthetic biology [12], optimisations and graphics [18] and synchronisation of distributed systems [10]. Some of the future challenges of the field are presented in [16]. The most up-to-date information on P systems can be found on its website [26].

Every membrane system consists of a set of *compartments* linked together in accordance with certain well-defined system structures, e.g., tree and graph in the case of cell P systems and tissue (or neural) P systems, respectively. Some systems have a static structure, others have a dynamic one. Each compartment contains a multiset of elements, either simple objects or more complex data, *strings*. These are either transformed or transferred between neighbour compartments, due to some *rules* which are specific to each compartment. A membrane system appears to be a computational model of a distributed system, where the structure of the system, the types of objects and transformations matter and collaborate in order to express a certain computation.

Membrane computing has been an umbrella for the proliferation of different variants of membrane systems. Not only studies investigating relationships between different classes of P systems, but also software tools supporting them have been considered. The best known tool that covers the most used P system models has a specification language, known generically as *P–Lingua* [25], which provides adequate syntax for each of the variants of P systems supported. P–Lingua aims to keep the syntax as close as possible to the original models and provides a simulation platform for all these models and a consistent user interface environment, called MeCoSim [24].

An alternative approach has been considered, by defining a more general membrane system model, allowing to relatively easily specify the most utilised P system models. This model is called *kernel P systems* (*kP systems*). A revised version of the model and the specification language can be found in [14] and its usage to specify the 3-colouring problem and a comparison to another solution provided in a similar context [9], is described in [15]. The kP systems have also been used to specify and analyse, through formal verification, synthetic biology systems [22, 21].

Kernel P systems are supported by a software framework, kPWorkbench [1, 2], which integrates a set of related simulation and verification methodologies and tools.

All these classes of P systems deal with non-deterministic behaviour, but in various circumstances, especially when biological systems are considered, stochastic systems are more appropriate. In this respect several variants of stochastic P systems have been introduced and utilised to model various problems in systems and synthetic biology [12]. A variant which is also supported by a simulation and verification environment is based on Gillespie approach for executing the system [17]. The tool and some applications are presented in [6].

In this work, we utilise kP systems and stochastic P systems (based on Gillespie approach), together with the corresponding software platforms developed, in order to model and verify certain properties of biological systems. The novelty of the approach is given by (a) the methodology that combines quantitative and qualitative analysis; (b) the modular way of specifying systems very close to the their informal descriptions; and (c) by the power of the verification method, relying on *model checking* techniques, which combines various approaches in order to adequately check the desired properties.

The chapter consists of five sections. Section 2 introduces the basic concepts related to kernel P systems and stochastic P systems. Section 3 introduces the AND and OR logic gates. Section 4 discusses the model described by using stochastic P systems and the modules associated with them. In Section 5 it is presented the verification methodology combining quantitative and qualitative analysis. Finally, Section 6 draws conclusions.

1.2 P Systems – Basic Definitions

The reader is assumed to be familiar with basic elements of membrane computing, e.g., from [28]. Some basic concepts utilised in the sequel will be introduced. Let A be an alphabet. An word with elements from A is a sequence containing these elements. The set of all words over A is denoted by A^*; λ denotes the empty word and $A^+ = A \setminus \{\lambda\}$. A multiset w over A is a mapping, $w : A \longrightarrow \mathbb{N}$ and $w(a)$, $a \in A$, defines the number of occurrences of a in the multiset. In the sequel a multiset will be defined by a word where the order of the elements is not considered.

In this section we will introduce two P system models, a non-deterministic version, called kernel P systems, and a stochastic one, called stochastic P systems.

1.2.1 Kernel P Systems

A kP system is made of compartments placed in a graph-like structure. Each compartment C_i, $1 \leq i \leq n$, has a type $t_i = (R_i, \sigma_i)$, $t_i \in T$, where T represents the set of all types, describing the associated set of rules R_i and the execution strategy, σ_i, of that compartment. In the sequel, we will present a simplified version of kP systems – for the full definition we refer to [13].

Definition 1. A *kernel P (kP) system* of degree n is a tuple

$$k\Pi = (A, \mu, C_1, \ldots, C_n, i_0),$$

where A is a finite set of elements called *objects*; μ defines the *membrane structure*, a graph, (V, E), where V are vertices indicating compartments, and E edges; $C_i = (t_i, w_i)$, $1 \leq i \leq n$, is a *compartment* of the system consisting of a *compartment type*, $t_i \in T$, and an *initial multiset*, w_i over A; i_0 is the *output compartment* where the result is obtained.

Each rule r may have a *guard* g denoted as $r\ \{g\}$. The rule r is applicable to a multiset w when its left hand side is contained into w and g is true for w. The guards are constructed using multisets over A and relational and Boolean operators. For example, rule $r : ac \to c\ \{\geq a^3 \wedge < b^5\}$ can be applied to the current multiset, $w = a^5b^4c$, as it includes the left hand side of r, i.e., ac and the guard condition is satisfied by w – there are at least 3 a's and no more than 5 b's.

In the sequel, we will present the types of rules utilised by kP systems. In the more general definition of such systems, see [13], there are two main types of rules, *rewriting and communication rules* and *structure changing rules*. The later set of rules is meant to be used when the structure of system, involving both compartments and links, is changing. In this work, we will be using only rewriting and communication rules and the definition below will deal with these types of rules.

Definition 2. A *rewriting and communication* rule has the form: $x \to y\ \{g\}$, in compartment l_i, where $x \in A^+$ and y has the form $y = (a_1, t_1) \dots (a_h, t_h)$, $h \geq 0$, $a_j \in A$ and t_j indicates a compartment type from T – see Definition 1 – with instance compartments linked to the current compartment; if a link does not exist (the two compartments are not in E) then the rule is not applied; if a target, t_j, refers to a compartment type that has more than one instance connected to l_i, then one of them will be non-deterministically chosen;

Each compartment has an execution strategy for its set of rules that can be defined as a sequence $\sigma = \sigma_1 \& \sigma_2 \& \dots \& \sigma_n$, where σ_i denotes an atomic component of the form:

- ϵ, means *empty* execution strategy – an analogue of a *skip* instruction;
- r, a rule from the set R_t (the set of rules associated with type t), describes the fact that *if r is applicable, then it is executed*; otherwise, the compartment terminates the execution thread for this particular computational step and thus, no further rule will be applied;
- $(r_1, \dots, r_n)$, with $r_i \in R_t, 1 \leq i \leq n$, describes a *non-deterministic choice within a set of rules*; one and only one applicable rule will be executed, if such a rule exists, otherwise this is simply skipped;
- $(r_1, \dots, r_n)^*$, with $r_i \in R_t, 1 \leq i \leq n$, indicates that the rules $\{r_1, \dots, r_n\}$ are executed *iteratively an arbitrary number of steps*;
- $(r_1, \dots, r_n)^\top, r_i \in R_t, 1 \leq i \leq n$, represents the maximally parallel execution of a set of rules. If no rules are applicable, then the execution proceeds to the subsequent atom in the chain.

We introduce now the concept of a *configuration* of a kP system of degree n as being an n-tuple $\mathcal{K} = (u_1, \dots, u_n)$, where u_i is a multiset in compartment C_i, $1 \leq i \leq n$. The initial configuration is $\mathcal{K}_0 = (w_1, \dots, w_n)$. Starting from $\mathcal{K}_0$ and using rules from $R_1, \dots, R_n$ in accordance with the execution strategies $\sigma_1, \dots, \sigma_n$, one gets a sequence of configurations. The process of getting a configuration from another one is called *transition*. A *computation* of Π is a sequence of transitions starting from $\mathcal{K}_0$. If the sequence is finite then this leads to a *halting computation* and the result is read out from i_0. In applications one can consider partial computations and the result is not always restricted to only one single compartment.

We present now an example of a kP system with two compartments by using first the notations introduced so far and then its transcription in a machine readable language, called *kP–Lingua* [11].

Example 1. There are two types $t_1 = (R_1, \sigma_1)$, and $t_2 = (R_2, \sigma_2)$, where
$R_1 = \{r_{1,1} : bb \to (a, t_2)\ \{\geq b^2\}; r_{1,2} : b \to (b, t_1)(b, t_1)\}$, $R_2 = \{r_{2,1} : a \to (c, t_1)(c, t_1)\}$ and
$\sigma_1 = (r_{1,1}, r_{1,2})$, $\sigma_2 = (r_{2,1})$. One can notice that rule $r_{1,1}$ has a guard that requires at least $2b$'s to be present in the current multiset. The execution strategies, σ_1 and σ_2, are non-deterministic choices. The kP system of degree 2 is given by

$$k\Pi_1 = (\{a, b, c, d\}, \mu, C_1, C_2, 1),$$

where μ is a graph with two vertices, C_1, C_2, and an edge between them. The two components are given by $C_1 = (t_1, w_1)$, $C_2 = (t_2, w_2)$, where $w_1 = d^2b$, $w_2 = d$. The initial configuration of the system is $\mathcal{K}_0 = (d^2b, d)$. The only possible transition allows only the rule $r_{1,2}$ to be applied in C_1 and consequently the next configuration is $\mathcal{K}_1 = (d^2b^2, d)$. In this configuration, one can use in C_1 either $r_{1,1}$ or $r_{1,2}$ (both are applicable) and nothing in C_2; hence, one gets either $\mathcal{K}'_2 = (d^2b^3, d)$ or $\mathcal{K}''_2 = (d^2, da)$, respectively. From $\mathcal{K}'_2$ one can continue with either $r_{1,1}$ or $r_{1,2}$ in C_1 and nothing in C_2. In $\mathcal{K}''_2$ one can only use $r_{2,1}$ in C_2 leading to $\mathcal{K}''_3 = (d^2c^2, d)$, which is a final configuration and we obtain a halting computation with the result in C_1, d^2c^2.

This example written in kP–Lingua is:

```
type t1 {
    choice {
        >= 2b : 2b ->  a(C2) .
         b ->  2b .
    }
}
type t2 {
    choice {
      a -> {2c}(C1) .
    }
}
C1 {2d, b} (t1) - C2 {d} (t2) .
```

Above, `t1, t2` denote two compartment types, which are instantiated as `C1, C2`, respectively. `C1` starts with the initial multiset `2d, b` and `C2` starts with `d`. The rules of `C1` are chosen non-deterministically, only one at a time – this is achieved by the use of the key word `choice`. The first rule is fired only when its guard becomes true. This rule also sends an `a` to the instance of `t2` that is linked. In the type `t2`, there is only one rule to be fired, which happens only when there is an `a` in the compartment `C2`.

1.2.2 Stochastic P Systems

In the case of stochastic P systems, constants are associated with rules in order to compute their probabilities. The precise definition is given below. It refers to a class of P systems, called *tissue P systems*, where the system structure is defined as a graph of components – a precise formal definition can be found in [28].

Definition 3. A *stochastic P (sP) system* is a model consisting of a tissue P system

$$sP = (O, L, \mu, M_1, \ldots, M_n, R_1, \ldots, R_n)$$

where O is a finite set of objects, called *alphabet*, denoting the entities involved in the system; L is a finite set of labels naming compartments; μ is a membrane structure composed of $n \geq 1$ membranes defining the regions or compartments of the system and their connections, forming an arbitrary graph; $M_i = (l_i, w_i)$, $1 \leq i \leq n$, is the initial configuration of the compartment or region defined by the membrane i, where $l_i \in L$ is the label of the compartment and $w_i \in O^*$ is a finite initial multiset of objects; $R_i = \{r_1^i, \ldots, r_{m_i}^i\}$, $1 \leq i \leq n$, is a set of multiset rewriting rules, of the form: $r_k^i : [x \rightarrow^{c_k} y]_{l_i}$, where x and y are multisets of objects (y might be empty) over O, representing the molecular species consumed and produced in the corresponding molecular interaction occurring in the compartment labelled l_i. An application of a rule of this form changes the content of the membrane with label l_i by replacing the multiset x with y. The stochastic constant c_k is used by the Gillespie algorithm [17] in order to compute the probabilities associated with the rules.

In each compartment of the sP system the execution strategy is based on Gillespie algorithm. Similarly to kP systems, one can define configurations, transitions and computations. Partial computations are also widely used in this context.

The model of the sP systems has been considered as the basis of the Infobiotics Workbench [6] where this is extended with some modularity features allowing a more flexible specification of a system. Each module has a name and some attributes associated with it. We do not provide the full formal definition of these modules (for a formal approach see [6, 31]), but we prefer to introduce them through some examples utilised later in this work.

The unregulated gene expression module, whereby some genes are expressed constitutively and independently of transcription factors is defined by the module:

$UnReg(\{G, R, P\}, \{c_1, c_2, c_3, c_4\})\{$
$G \rightarrow^{c_1} G + R;$
$R \rightarrow^{c_2} R + P;$
$R \rightarrow^{c_3};$
$P \rightarrow^{c_4}\}$

This module describes the process of transcribing the gene G into its corresponding mRNA, R, which in turn is translated into a protein P. The mRNA and the protein can be degraded. The propensities of these processes are determined by the stochastic coefficients c_i, $1 \leq i \leq 4$. Some variants of this might appear when more than a protein is involved. The module has the form:

$UnRegM(\{G, R, P_1, P_2\}, \{c_1, c_2, c_3, c_4, c_5, c_6\})\{$
$G \rightarrow^{c_1} G + R;$
$R \rightarrow^{c_2} R + P_1;$
$R \rightarrow^{c_3} R + P_2;$
$R \rightarrow^{c_4};$
$P_1 \rightarrow^{c_5};$
$P_2 \rightarrow^{c_6}\}$

When it is assumed that the protein is obtained in one step from the gene, the module is:

$UnRegS(\{G, P\}, \{c_1, c_2\})\{$
$G \rightarrow^{c_1} G + P;$
$P \rightarrow^{c_2}\}$

One can also describe the process of complex formation, when two molecules M_1 and M_2 form a more complex molecule $M_1.M_2$ and this might be reversible. The module is:

$Comp(\{M_1, M_2\}, \{c_1\})\{$
$M_1 + M_2 \rightarrow^{c_1} M_1.M_2\}$

A negative regulation of a gene, when a repressor protein R binds reversibly to the gene G preventing it to produce any protein, is given by:

$Neg(\{R, G\}, \{c_1, c_2\})\{$
$R + G \rightarrow^{c_1} R.G;$
$R.G \rightarrow^{c_2} R + G\}$

1.2.3 Tools

The specifications written in kP–Lingua are supported by a software platform, kPWorkbench, which integrates a set of tools and translators that bridge several target specifications that we employ for kP system models, written in kP-Lingua. kPWorkbench permits *simulation* and *formal verification* of kP system models using several simulation and verification methodologies and tools.

The Infobiotics Workbench (IBW) is an integrated software suite of tools to perform in silico experiments for sP models in systems and synthetic biology [6]. This software platform includes *simulators* and tools for *formal verification* of stochastic models.

The IBW tool is aimed at providing support for quantitative (stochastic) analysis of systems, whereas kPWorkbench is meant to help with qualitative analysis. In this respect, systems are specified within IBW, by using modularity features and then analysed with the existing tools of this environment. For qualitative analysis these are then automatically translated into a non-deterministic version of the system which is then analysed within the kPWorkbench environment.

1.3 Genetic Logic Gates

Genetic logic gates have been considered in various papers, including [4, 34, 30], using various synthetic biology tools, amongst them GEC [29], Eugene [5] and Proto [3]. In [33, 23], we have studied two basic logic gates, AND and OR, using the IBW tool for quantitative analysis and kPWorkbench for qualitative one. Here, we provide a summary of our results.

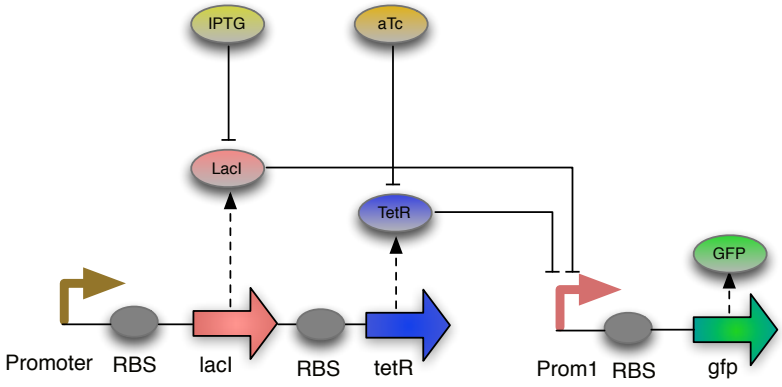
The genetic parts and designs of these gates are proposed by Beal et al. [4]. Both gates use two inducers, `aTc` and `IPTG`, as input and use `GFP` as output. `aTc` and `IPTG` disable the activities of `TetR` and `LacI` proteins, respectively.

Figure 1.1a illustrates the genetic design of an AND gate, which receives two input signals: `aTc` and `IPTG`. In this system, the transcription factors `LacI` and `TetR` are expressed by a gene controlled by the same promoter. The `aTc` molecules repress `TetR`, and `IPTG` molecules repress `LacI`, to prevent them from inhibiting the production of `GFP` by binding to the corresponding promoter which up-regulates the expression of `GFP`. If both `IPTG` and `aTc` are set to high, then neither `LacI` nor `TetR` can inhibit the `GFP` production.

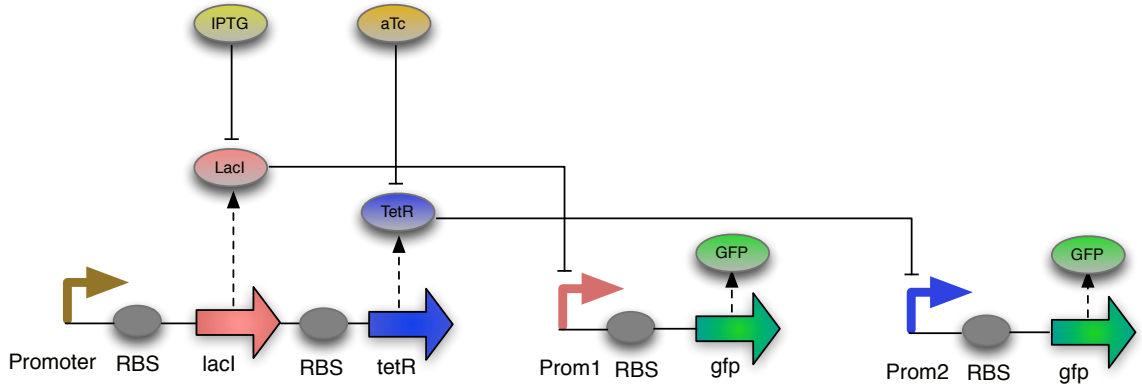
Figure 1.1b illustrates a genetic OR gate, comprising two mechanisms. Each mechanism leads to the production of `GFP`, when it is activated. The first mechanism is repressed by `LacI` while the second is repressed by `TetR`. Therefore, `GFP` can be produced from the former when `IPTG` is set to high and from the latter when `aTc` is set to high.

The stochastic model, in each of the two cases, consists of an sP system with one compartment and a set of stochastic rules, governing the kinetic and stochastic behaviour of the system. The rewriting rules and the kinetic constants (taken from [4]) of the devices are described in the Tables (a) AND & (b) OR of the Annex.

In the next section we will show how the two gates are modelled by using the modular approach provided by the sP systems and available as part of the IBW specification component.



(a) AND gate



(b) OR gate

Fig. 1.1: Beal et. al.'s genetic devices functioning as an AND and OR gate (taken from [33]).

1.4 Modelling with sP Systems

The sP systems model utilised for each of the two logic gates consists of a one compartment system and an alphabet of objects including all the species and molecules that appear in the two sets of reactions listed in Tables (a) AND & (b) OR of the Annex. For expressing the behaviour of each of these systems we will be using modules as they are supported by the IBW environment. These specifications are also automatically translated into non-deterministic kP systems and made availale to kPWorkbench.

The AND logic gate fully described by the reactions listed in Table (a) AND can be specified using modules in a manner that maps better the informal specification above. The rules r_1, r_2, r_3 and r_{10}, r_{11}, r_{12} can be embedded into a module $UnRegM$ which expresses the fact that the transcription factors `LacI` and `TetR` are expressed by a gene, but also that the mRNA and the transcription factors

degrade. Each of the reactions r_4 and r_5 expresses a complex formation, whereby the `aTc` molecules repress `TetR`, and `IPTG` molecules repress `LacI`, respectively. These are captured by *Comp* modules. The fact that `LacI` and `TetR` inhibit the `GFP` production (rules r_{6a}, r_{6b} and r_{7a}, r_{7b}) is captured by two modules describing this negative regulation, *Neg*. Finally, the `GFP` production (rules r_8, r_9) is defined by a module *UnRegS*. The complete specification of the AND logic gate using modules is:

$UnRegM(\{\texttt{gene\_LacI\_TetR}, \texttt{mLacI\_TetR}, \texttt{LacI}, \texttt{TetR}\}, \{k_1, k_2, k_3, k_{12}, k_{10}, k_{11}\})$
$Comp(\{\texttt{LacI}, \texttt{IPTG}\}, \{k_4\})$
$Comp(\{\texttt{TetR}, \texttt{aTc}\}, \{k_5\})$
$Neg(\{\texttt{LacI}, \texttt{gene\_GFP}\}, \{k_{6a}, k_{6b}\})$
$Neg(\{\texttt{TetR}, \texttt{gene\_GFP}\}, \{k_{7a}, k_{7b}\})$
$UnRegS(\{\texttt{gene\_GFP}, \texttt{GFP}\}, \{k_8, k_9\})$

The OR logic gate is very similar to AND with respect to modelling modules. The first three lines, using modules *UnRegM* and *Comp* twice are the same. As the OR gate uses two mechanisms to produce GFP then the two *Neg* modules of the AND gate are replaced by

$Neg(\{\texttt{LacI}, \texttt{gene\_GFP1}\}, \{k_{6a}, k_{6b}\})$
$Neg(\{\texttt{TetR}, \texttt{gene\_GFP2}\}, \{k_{7a}, k_{7b}\})$

and finally, the *UnRegS* of the AND gate is replaced by two *UnRegS* modules responsible for producing `GFP`.

Having the systems implemented by using modules one can run various simulations with IBW environment and check their behaviours. The trajectories of both gate dynamics for the four different input combinations of low and high `aTc` and `IPTG` concentrations are shown in Figure 1.2. The graphs presented show that the gates quickly approach a steady state with output concentrations. These show that the models implement the desired Boolean logics. Apart from simulations revealing various aspects of the systems' behaviour the tools discussed earlier provide more insights into the system by revealing certain properties or relationships between different components. All these will be investigated in the next section.

The translation of the sP system into a kP system is obtained automatically by removing the kinetic coefficients of the former. The kP system obtained can be executed in kPWorkbench by using the execution strategy corresponding to non-deterministic choice. The simulation of such a non-deterministic system does not bring any new information about the model, but this specification is useful for the formal verification performed in the next section.

1.5 P Systems Verification

In this section we briefly present a methodology for verifying P system models using model checking approaches. This has been developed by looking at quantitative and qualitative results whereby various model checking tools have been used to investigate properties of different types [22, 20]. In this work we will be illustrating the use of two model checkers, Prism [19] for quantitative analysis and NuSMV [7] for qualitative aspects; they are part of the IBW and kPWorkbench platforms, respectively. They

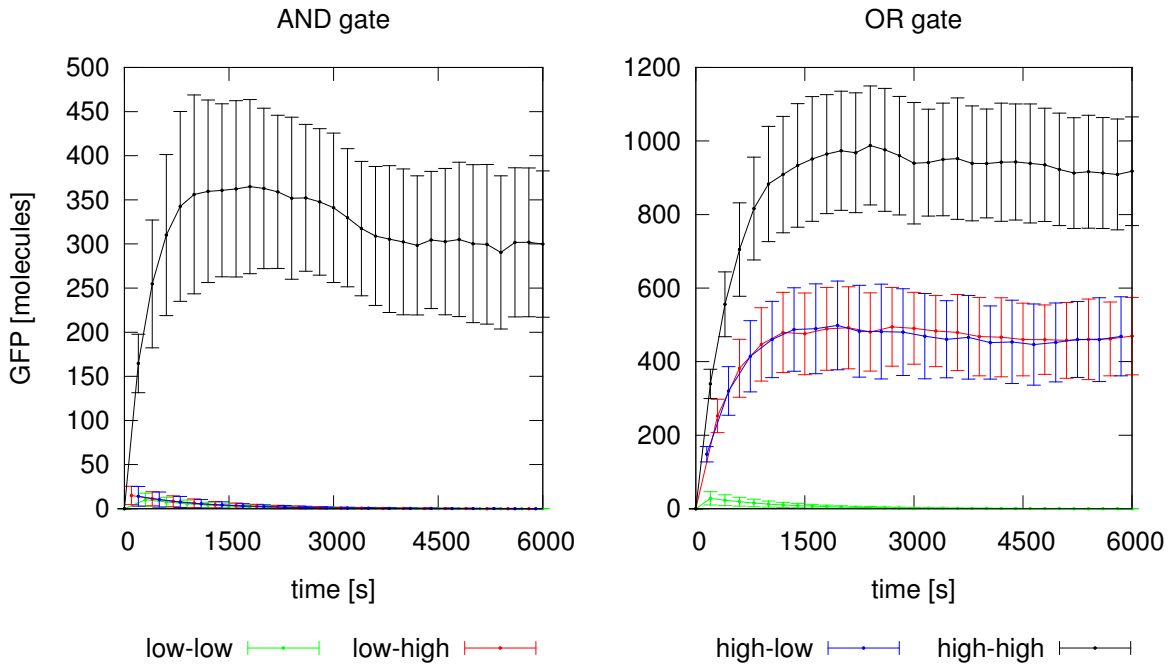


Fig. 1.2: GFP expression in the AND (left) and (OR) gate over time for the aTc/IPTG input combinations *low-low*, *low-high*, *high-low*, and *high-high* (taken from [33]). Error bars denote the standard deviations of 100 statistically independent samples.

can be directly called from these platforms and the queries can be formulated in a natural language format [6].

In standard logic gates, any voltage value above a threshold value, such as 3V, is considered as 1. Since there is no such a standard value for genetic Boolean gates, we propose a threshold for this particular design. To analyse the behaviour of the genetic devices formally, we verify the following property using PRISM:

"What is the likelihood that GFP *eventually exceeds the threshold* Thr *at time* t*?"*

which is expressed in a probabilistic logic (CSL) as

$$\mathrm{P}_{=?}\left[\mathrm{F}^{t}\ \mathtt{GFP} \geq Thr\right].$$

This query returns different results for different parameter values. For example, if we consider the OR gate, it returns 1 for $Thr = 100$, $t = 1000$, and $\mathtt{aTc} = \mathtt{IPTG} = 1000$.

When models are built, especially when these are complex, with many species and interactions, it is essential that one can verify their correctness. In many cases there are chains of reactions leading to certain results and it is important to check dependabilities and the way they influence certain results. In our case, both logic gates show certain dependabilities. For instance, one can check that GFP is not

present initially in the system, but it will eventually appear. One can verify whether GFP will finally appear in the system, by using the following statement in NuSMV:

"There are pathways in the system that eventually lead to the production of GFP"

$$\text{EF } (\texttt{GFP} > 0).$$

This property, expressed in CTL, is true, as expected.

We now show how one can check that a certain sequence of events must or might appear in a chain of reactions. This is illustrated by the relationship between the production of LacI and TetR and the complex formation of LacI.IPTG and TetR.aTc, respectively. We illustrate now the chain of events triggered by LacI by using a CTL formula in NuSMV:

"Always the LacI production might eventually lead to the complex LacI.IPTG."

$$\text{AG } (\texttt{LacI} > 0 \Rightarrow \text{EF } \texttt{LacI.IPTG} > 0)$$

This property is true, as expected.

1.6 Conclusions

In this paper, we have shown how an unconventional computing paradigm, membrane systems, is utilised to model and analyse various systems, especially biological systems. In particular, we have considered kP systems and stochastic P systems, together with the corresponding software platforms developed, in order to model and verify certain properties of biological systems.

Our approach is novel in the sense that our methodology (i) combines quantitative and qualitative analysis; (ii) is a modular way of specifying systems, (iii) employs simulation methods to analyse system dynamics and (iv) integrates various verification methods to adequately check the desired properties.

We are currently working on the next versions of IBW and kPWorkbench tools by incorporating more methods for specifying, modelling, simulating and verifying biological systems.

Acknowledgements.

MG and SK acknowledge the support provided for synthetic biology research by EPSRC ROADBLOCK (project number: EP/I031812/1). The work of FI was supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI (project number: PN-II-ID-PCE-2011-3-0688).

References

1. M. E. Bakir, F. Ipate, S. Konur, L. Mierlă, and I. Niculescu. Extended simulation and verification platform for kernel P systems. In *15th International Conference on Membrane Computing*, volume 8961 of *LNCS*, pages 158–168. Springer, 2014.

2. M. E. Bakir, S. Konur, M. Gheorghe, I. Niculescu, and F. Ipate. High performance simulations of kernel P systems. In *Proceedings of the 2014 IEEE 16th International Conference on High Performance Computing and Communication*, HPCC '14, pages 409–412, Paris, France, 2014.
3. J. Beal, T. Lu, and R. Weiss. Automatic compilation from high-level biologically-oriented programming language to genetic regulatory networks. *PLoS ONE*, 6(8):e22490, 2011.
4. J. Beal, A. Phillips, D. Densmore, and Y. Cai. High-level programming languages for biomolecular systems. In *Design and Analysis of Biomolecular Circuits*, pages 225–252. Springer, New York, 2011.
5. L. Bilitchenko, A. Liu, S. Cheung, E. Weeding, B. Xia, M. Leguia, J. C. Anderson, and D. Densmore. Eugene - a domain specific language for specifying and constraining synthetic biological parts, devices, and systems. *PLoS ONE*, 6(4):e18882, 2011.
6. J. Blakes, J. Twycross, S. Konur, F. Romero-Campero, N. Krasnogor, and M. Gheorghe. Infobiotics workbench: A P systems based tool for systems and synthetic biology. In *Applications of Membrane Computing in Systems and Synthetic Biology*, volume 7 of *Emergence, Complexity and Computation*, pages 1–41. Springer, 2014.
7. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV version 2: An open source tool for symbolic model checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, pages 359–364, Copenhagen, Denmark, 2002. Springer.
8. G. Ciobanu, Gh.. Păun, and M. J. Pérez-Jiménez, editors. *Applications of Membrane Computing*. Springer, 2006.
9. D. Díaz-Pernil, M. A. Gutiérrez-Naranjo, and M. J. Pérez-Jiménez. A uniform family of tissue P systems with cell division solving 3-COL in a linear time. *Theoretical Computer Science*, 404:76–87, 2008.
10. M. J. Dinneen, K. Yun-Bum, and R. Nicolescu. Faster synchronization in P systems. *Natural Computing*, 11(4):637–651, 2012.
11. C. Dragomir, F. Ipate, S. Konur, R. Lefticaru, and L. Mierlă. Model checking kernel P systems. In *14th International Conference on Membrane Computing*, volume 8340 of *LNCS*, pages 151–172. Springer, 2013.
12. P. Frisco, M. Gheorghe, and M. J. Pérez-Jiménez, editors. *Applications of Membrane Computing in Systems and Synthetic Biology*. Springer, 2014.
13. M. Gheorghe, F. Ipate, and C. Dragomir. Kernel P systems. In *10th Brainstorming Week on Membrane Computing*, pages 153–170. Fénix Editora, 2012.
14. M. Gheorghe, F. Ipate, C. Dragomir, L. Mierlă, L. Valencia-Cabrera, M. García-Quismondo, and M. J. Pérez-Jiménez. Kernel P systems - version 1. In *11th Brainstorming Week on Membrane Computing*, pages 97–124. Fénix Editora, 2013.
15. M. Gheorghe, F. Ipate, R. Lefticaru, M. J. Pérez-Jiménez, A. Ţurcanu, L. Valencia-Cabrera, M. García-Quismondo, and L. Mierlă. 3-Col problem modelling using simple kernel P systems. *Int. Journal of Computer Mathematics*, 90(4):816–830, 2012.
16. M. Gheorghe, G. Păun, M. J. Pérez-Jiménez, and G. Rozenberg. Research frontiers of membrane computing: Open problems and research topics. *International Journal of Foundations of Computer Scence*, 24:547–624, 2013.
17. D. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403–434, 1976.
18. G. L. Gimel'farb, R. Nicolescu, and S. Ragavan. P system implementation of dynamic programming stereo. *Journal of Mathematical Imaging and Vision*, 47(1–2):13–26, 2013.
19. A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. TACAS*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.
20. S. Konur and M. Gheorghe. Property-driven methodology for formal analysis of synthetic biology systems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 12(2):360–371, 2015.
21. S. Konur, M. Gheorghe, C. Dragomir, F. Ipate, and N. Krasnogor. Conventional verification for unconventional computing: a genetic XOR gate example. *Fundamenta Informaticae*, 134(1-2):97–110, 2014.
22. S. Konur, M. Gheorghe, C. Dragomir, L. Mierlă, F. Ipate, and N. Krasnogor. Qualitative and quantitative analysis of systems and synthetic biology constructs using P systems. *ACS Synthetic Biology*, 4(1):83–92, 2015.
23. S. Konur, C. Ladroue, H. Fellermann, D. Sanassy, L. Mierla, F. Ipate, S. Kalvala, M. Gheorghe, and N. Krasnogor. Modeling and analysis of genetic boolean gates using Infobiotics Workbench. In *Verification of Engineered Molecular Devices and Programs*, pages 26–37, Vienna, Austria, 2014.
24. MeCoSim website. url: `http://www.p-lingua.org/mecosim/`.
25. P-Lingua website. url: `http://www.p-lingua.org`.
26. P systems website. url: `http://ppage.psystems.eu`.
27. G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
28. G. Păun, G. Rozenberg, and A. Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.

29. M. Pedersen and A. Phillips. Towards programming languages for genetic engineering of living cells. *Journal of The Royal Society Interface*, 6(Suppl 4):S437–S450, 2009.
30. S. Regot, J. Macia, N. Conde, K. Furukawa, J. Kjellen, T. Peeters, S. Hohmann, E. de Nadal, F. Posas, and R. Sole. Distributed biological computation with multicellular engineered networks. *Nature*, 469(7329):207–211, 2011.
31. F. Romero-Campero, J. Twycross, M. Càmara, M. Bennett, M. Gheorghe, and N. Krasnogor. Modular assembly of cell systems biology models using P systems. *International Journal of Foundations of Computer Science*, 20(3):427–442, 2009.
32. G. Rozenberg, T. Bäck, and J. N. Kok, editors. *Handbook of Natural Computing*. Springer, 2012.
33. D. Sanassy, H. Fellermann, N. Krasnogor, S. Konur, L. Mierlă, M. Gheorghe, C. Ladroue, and S. Kalvala. Modelling and stochastic simulation of synthetic biological boolean gates. In *Proceedings of the 2014 IEEE 16th International Conference on High Performance Computing and Communication*, HPCC '14, pages 404–408, Paris, France, 2014.
34. A. Tamsir, J. J. Tabor, and C. A. Voigt. Robust multicellular computing using genetically encoded NOR gates and chemical 'wires'. *Nature*, 469(7329):212–215, 2011.

Annex. Tables

(a) AND gate by Beal et al. (taken from [33])

r_{0a} : $\overset{k_{0a}}{\rightarrow}$ IPTG	$k_{0a} \in \{0, 1000\}$
r_{0b} : $\overset{k_{0b}}{\rightarrow}$ aTc	$k_{0b} \in \{0, 1000\}$
r_1 : gene_LacI_TetR $\overset{k_1}{\rightarrow}$ gene_LacI_TetR + mLacI_TetR	$k_1 = 0.12$
r_2 : mLacI_TetR $\overset{k_2}{\rightarrow}$ mLacI_TetR + LacI	$k_2 = 0.1$
r_3 : mLacI_TetR $\overset{k_3}{\rightarrow}$ mLacI_TetR + TetR	$k_3 = 0.1$
r_4 : LacI + IPTG $\overset{k_4}{\rightarrow}$ LacI-IPTG	$k_4 = 1.0$
r_5 : TetR + aTc $\overset{k_5}{\rightarrow}$ TetR-aTc	$k_5 = 1.0$
r_{6a} : gene_GFP + LacI $\overset{k_{6a}}{\rightarrow}$ gene_GFP-LacI	$k_{6a} = 1.0$
r_{6b} : gene_GFP-LacI $\overset{k_{6b}}{\rightarrow}$ gene_GFP + LacI	$k_{6b} = 0.01$
r_{7a} : gene_GFP + TetR $\overset{k_{7a}}{\rightarrow}$ gene_GFP-TetR	$k_{7a} = 1.0$
r_{7b} : gene_GFP-TetR $\overset{k_{7b}}{\rightarrow}$ gene_GFP + TetR	$k_{7b} = 0.01$
r_8 : gene_GFP $\overset{k_8}{\rightarrow}$ gene_GFP + GFP	$k_8 = 1.0$
r_9 : GFP $\overset{k_9}{\rightarrow}$	$k_9 = 0.001$
r_{10} : LacI $\overset{k_{10}}{\rightarrow}$	$k_{10} = 0.01$
r_{11} : TetR $\overset{k_{11}}{\rightarrow}$	$k_{11} = 0.01$
r_{12} : mLacI_TetR $\overset{k_{12}}{\rightarrow}$	$k_{12} = 0.001$

(b) OR gate by Beal et al. (taken from [33])

$r_0 - r_5$	same as the rules $r_0 - r_5$ of the AND gate above	
r_{6a} :	gene_GFP1 + LacI $\overset{k_{6a}}{\rightarrow}$ gene_GFP1-LacI	$k_{6a} = 1.0$
r_{6b} :	gene_GFP1-LacI $\overset{k_{6b}}{\rightarrow}$ gene_GFP1 + LacI	$k_{6b} = 0.01$
r_{7a} :	gene_GFP2 + TetR $\overset{k_{7a}}{\rightarrow}$ gene_GFP2-TetR	$k_{7a} = 1.0$
r_{7b} :	gene_GFP2-TetR $\overset{k_{7b}}{\rightarrow}$ gene_GFP2 + TetR	$k_{7b} = 0.01$
r_8 :	gene_GFP1 $\overset{k_8}{\rightarrow}$ gene_GFP1 + GFP	$k_8 = 1.0$
r_9 :	gene_GFP2 $\overset{k_9}{\rightarrow}$ gene_GFP2 + GFP	$k_9 = 1.0$
$r_{10} - r_{13}$	same as the rules $r_9 - r_{12}$ of the AND gate	