A Hybrid Approach to Modelling Biological Systems

Francesco Bernardini[1], Marian Gheorghe[2],
Francisco José Romero-Campero[3], Neil Walkinshaw[2]

[1] Leiden Institute of Advanced Computer Science
Leiden University
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
bernardi@liacs.nl
[2] Department of Computer Science
The University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
m.gheorghe@dcs.shef.ac.uk, n.walkinshaw@dcs.shef.ac.uk
[3] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
fran@us.es

Summary. This paper investigates a hybrid approach to modelling molecular interactions in biology. Some computer science models are presented, namely, P systems, π-calculus and Petri nets, and two tools, Daikon, used initially in reverse-engineering, and PRISM, a probabilistic model checker. All these approaches are investigated for their complementary role in modelling which is illustrated through a simple case study.

1 Introduction

In the last decade there has been a great interest in using theoretical computer science models in biology, based on different paradigms (process algebras, cellular automata, Lindenmayer systems, Petri nets, Boolean functions, P systems, etc.) with the aim of providing an understandable, extensible and computable modelling framework while keeping the needed formalisation to perform mathematical analysis. Every such model covers certain aspects of a system and combining two or more leads to obtaining a better and more powerful modelling approach. In order to include quantitative and qualitative aspects, there have been suggested various variants of certain models with new features like: Petri nets [10, 22], stochastic π-calculus [28] and stochastic P systems [19].

In this paper we investigate the concerted use of different methods that will reveal a new vision on modelling biological systems by combining different comple-

mentary approaches. This is quite different from the hybrid approach discussed by [1] where it is shown how to switch between deterministic and stochastic behaviour.

Section 2 introduces the three modelling approaches used in the paper: P systems, *pi*-calculus and Petri nets, as well as Daikon tool and a simple example involving a regulatory network that will be modelled within each approach. Section 3 presents Daikon's findings and the analysis of the invariants provided. The following two sections show how PRISM and a Petri net tool are used in order to confirm some of the properties suggested by Daikon analysis. The final section summarises our findings.

2 Modelling Paradigms

In this section we will present three modelling approaches, namely P systems, π-calculus, Petri nets. A simple case study will be used to illustrate the approach. This example will be written directly into the three modelling paradigms mentioned above, P systems, π-calculus and Petri nets and will be executed with a simulator dedicated to P systems. In this way we will show how the same problem is modelled with different paradigms and consequently will be able to point out to specific characteristics of these approaches that are used for the same purpose. A system of differential equations will be also associated to this example and the results obtained will be compared to the stochastic behaviour exhibited by the P system simulator. In the next three sections we will be using a tool called Daikon to reveal certain properties of our models as they appear through data sets generated by simulators, and two tools, namely PRISM and PIPE, that are used to analyse and verify properties identified by Daikon.

The aim of this investigation is not to study the relationships between the results produced by using differential equations and those generated by P systems. This has been already considered for a special class of P systems working in a deterministic manner according to a metabolic algorithm [7]. In this study we will be using differential equations only as a substitute for real data in order to illustrate our approach that allows us to "guess" certain properties of the model and then to verify whether they hold or not as general properties or just only happens to be true for the instances generated by simulation.

Nowadays ordinary differential equations (ODE) constitute the most widely used approach in modelling molecular interaction networks in cell systems. They have been used successfully to model kinetics of conventional macroscopic chemical reactions. Nevertheless the realisation of a reaction network as a system of ODEs is based on two assumptions. First, cells are assumed to be well stirred and homogeneous volumes so that concentrations do not change with respect to space. Whether or not this is a good approximation depends on the time and space scales involved. In bacteria it has been shown that molecular diffusion is sufficiently fast to mix proteins. This is not the case in eukaryotic cells where the volume is considerably bigger and it is structured in different compartments like

nucleus, mitochondria, golgi body, etc. The second basic assumption is that chemical concentrations vary continuously over time. This assumption is valid if the number of molecules of each species in the reaction volume (the cell or the subcellular compartment) are sufficiently large and the reactions are fast. A sufficiently large number of molecules is considered to be at least thousands of molecules; for hundreds or fewer molecules the continuous approach is questionable.

Writing and solving numerically a system of ODE describing a chemical reaction network can be largely automated. Each species is assigned a single variable $X(t)$ which represents the concentration of the species at time t. Then, for each molecular species, a differential equation is written to describe its concentration change over time due to interactions with other species in the system. The rate of each reaction is represented using a kinetic rate law, which commonly depends on one or more rate constants. Exponential decay law, mass action law and Michaelis-Menten dynamic are the most widely used kinetic mechanisms. Finally in order to solve the system of ODEs we must impose a set of initial condition representing the initial concentration of the species.

Due to the limitations of ODEs to handle cellular systems with low number of molecules and spatial heterogeneity, some computational approaches have recently been proposed. In what follows we disccuss three different approaches, P systems, π-calculus and Petri nets.

2.1 P systems

Membrane computing is an emergent branch of natural computing introduced by Gh. Păun in [18]. The models defined in this context are called P systems. In the sequel we will use membrane computing and P systems with the same meaning. Roughly speaking, a P system consists of a cell-like membrane structure, in the compartments of which one places multisets of objects and strings which evolve according to given rules. Recently P systems have been used to model biological phenomena within the framework of computational systems biology presenting models of oscillatory systems [6], signal transduction [19], gene regulation control [20], quorum sensing [27] and metapopulations [21]. In this respect, P systems present a formal framework for the specification and simulation of cellular systems which integrates structural and dynamic aspects in a comprehensive and relevant way while providing the required formalisation to perform mathematical and computational analysis.

In the original approach of P systems the rules are applied in a maximally parallel way. This produces two inaccuracies: the reactions represented by the rules do not take place at the correct rate, and all time steps are equal and do not represent the time evolution of the real system. In order to solve these two problems stochastic P systems were introduced in [19].

Definition 1. *A stochastic P system is a construct*

$$\Pi = (O, L, \mu, M_1, M_2, \ldots, M_n, R_1, \ldots, R_n)$$

where:

- *O is a finite alphabet of symbols representing objects;*
- *L 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 L;*
- *$M_i = (l_i, w_i, s_i)$, for each $1 \leq i \leq n$, is the initial configuration of membrane i with $l_i \in L$, the label of this membrane, $w_i \in O^*$ a finite multiset of objects and s_i a finite set of strings over O;*
- *R_i, for each $1 \leq i \leq n$, is a finite set of rewriting rules associated with membrane i, of one of the following two forms:*
  - *Multiset rewriting rules:*

$$obj_1\,[\,obj_2\,]_l \xrightarrow{k} obj_1'\,[\,obj_2'\,]_l \tag{1}$$

with $obj_1, obj_2, obj_1', obj_2' \in O^*$ some finite multisets of objects and l a label from L. A multiset of objects, obj is represented as $obj = o_1 + o_2 + \cdots + o_m$ with $o_1, \ldots, o_m \in O$.
These rules are multiset rewriting rules that operate on both sides of membranes, that is, a multiset obj_1 placed outside a membrane labelled by l and a multiset obj_2 placed inside the same membrane can be simultaneously replaced with a multiset obj_1' and a multiset obj_2', respectively.

  - *String rewriting rules:*

$$[\,obj_1 + str_1; \ldots; obj_p + str_p\,]_l \xrightarrow{k}$$
$$[\,obj_1' + str_{1,1}' + \ldots str_{1,i_1}'; \ldots; obj_p' + str_{p,1}' + \ldots str_{p,i_p}'\,]_l \tag{2}$$

A string str is represented as follows $str = \langle s_1.s_2.\cdots.s_i \rangle$ where $s_1, \ldots, s_i \in O$. In this case each multiset of objects obj_j and string str_j, $1 \leq j \leq p$, are replaced by a multiset of objects obj_j' and strings $str_{j,1}' \ldots str_{j,i_j}'$.
The stochastic constant k is used to compute the propensity of the rule by multiplying it by the number of distinct possible combinations of the objects and substrings present on the left-side of the rule with respect to the current contents of membranes involved in the rule. The propensity associated with each rule is used to compute the probability and time needed to apply it.

Cellular systems consisting of molecular interactions taking place in different locations of living cells are specified using stochastic P systems as follows. Different regions and compartments are specified using membranes. Each molecular species is represented by an object in the multiset associated with the region or compartment where the molecule is located. The multiplicity of each object represents the number of molecules of the molecular species represented by the object. Strings are used to specify the genetic information encoded in DNA and RNA. Molecular

interactions, compartment translocation and gene expression are specified using rewriting rules on multisets of objects and strings - see Table 1.

In stochastic P systems [19] constants are associated with rules in order to compute their probabilities and time needed to be applied according to Gillespie algorithm. This approach is based on a Monte Carlo algorithm for stochastic simulation of molecular interactions taking place inside a single volume [8]. In contrast to this, in P systems we have a membrane structure delimiting different compartments (volumes), each one with its own set of rules (molecular interactions) and multiset of objects and strings (molecules). In this respect, a scheduling algorithm called the Multicompartmental Gillespie algorithm [19] is used so that each compartment evolves according to a different Gillespie algorithm. In this point our approach differs from other computational approaches which run a single Gillespie algorithm across the whole system without taking into account the compartmentalised cellular structure [10, 28].

Biochemistry	**P System**
Compartment	Region defined by a membrane
Molecule	Object
Molecular Population	Multiset of objects
Biochemical Transformation	Rewriting rule
Compartment Translocation	Boundary rule

Table 1. Modelling Principles in P Systems

We illustrate our approach with a biomolecular system consisting in positive, negative and constitutive expression of a gene. Our model includes the specification of a gene, its transcribed RNA, the corresponding translated protein and activator and repressor molecules which bind to the gene producing an increase in transcription rate or prevent the gene from being transcribed, respectively. The bacterium where the system is located is represented using a membrane. The stochastic constants used in our model are taken from the gene control system in the lac operon in E. coli [2, 13, 14]. In this case transcription and translation have been represented using rewriting rules on multisets of objects, a more detailed description of the concurrent processes of transcription and translation using rewriting rules on strings is presented in [20]. The P systems model is formally defined in Figure 1. It consists of one single compartment labelled b, with no strings, and consequently using only multiset rewriting rules. The model refers to three distinct initial conditions, denoted by multisets $M_{0,i}$, and corresponding to constitutive expression, positive and negative regulations, respectively. Simulations of constitutive expression and positive regulation case studies are presented in Figure 2 using a tool available at [30]. A set of ordinary differential equations and their associated graphs, modelling the same examples, are provided in Figure 3. The ODE model is not used here to show its relationship to the previous P systems

approach, but to provide a set of data that normally is taken through biological experiments. This will only be used to provide data measurements that will help identifying and validating properties of the P systems model.

$\Pi = (\{gene, rna, protein, act, rep, act\text{-}gene, rep\text{-}gene\}, \{b\}, [\ ]_b, (b, M_i, \emptyset), \{r_1, \ldots, r_9\})$

Initial multisets: $M_{0,1} = gene$; $M_{0,2} = gene + act... + act$ and $M_{0,3} = gene + rep... + rep$ where act and rep occur 10 times each.
Rules:

$r_1: \ [\ gene\ ]_b \xrightarrow{c_1} [\ gene + rna\ ]_b \quad c_1 = 0.347\ min^{-1}$
$r_2: \ [\ rna\ ]_b \xrightarrow{c_2} [\ rna + protein\ ]_b \quad c_2 = 0.174\ min^{-1}$
$r_3: \ [\ rna\ ]_b \xrightarrow{c_3} [\ ]_b \quad c_3 = 0.347\ min^{-1}$
$r_4: \ [\ protein\ ]_b \xrightarrow{c_4} [\ ]_b \quad c_4 = 0.0116\ min^{-1}$
$r_5: \ [\ act + gene\ ]_b \xrightarrow{c_5} [\ act\text{-}gene\ ]_b \quad c_5 = 6.6412087\ molec^{-1} min^{-1}$
$r_6: \ [\ act\text{-}gene\ ]_b \xrightarrow{c_6} [\ act + gene\ ]_b \quad c_6 = 0.6\ s^{-1}$
$r_7: \ [\ act\text{-}gene\ ]_b \xrightarrow{c_7} [\ act\text{-}gene + rna\ ]_b \quad c_7 = 3.47\ min^{-1}$
$r_8: \ [\ rep + gene\ ]_b \xrightarrow{c_8} [\ rep\text{-}gene\ ]_b \quad c_8 = 6.6412087\ molec^{-1} min^{-1}$
$r_9: \ [\ rep\text{-}gene\ ]_b \xrightarrow{c_9} [\ rep + gene\ ]_b \quad c_9 = 0.6\ min^{-1}$

Fig. 1. P system model of gene expression.

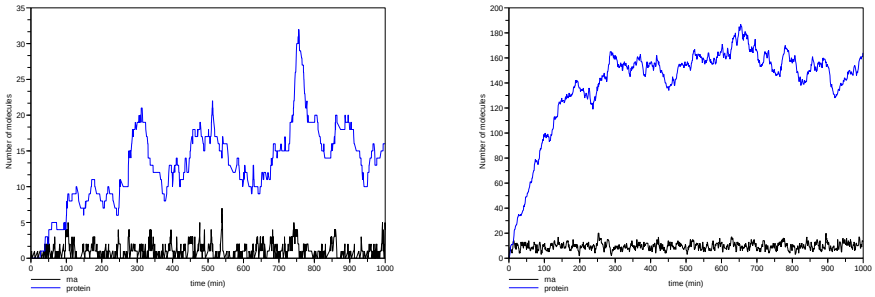


Fig. 2. Constitutive expression and positive regulation.

2.2 π-calculus

The π-calculus approach was introduced as a formal language to describe mobile concurrent processes [17]. It is now a widely accepted model for interacting systems with dynamically evolving communication topology. The π-calculus has a

$$\frac{dr}{dt} = c_1 - c_3 r + c_7 \frac{act}{act + K} \quad \text{where } K \text{ is the Michaelis-Menten constant}$$
$$\frac{dp}{dt} = c_2 r - c_4 p$$

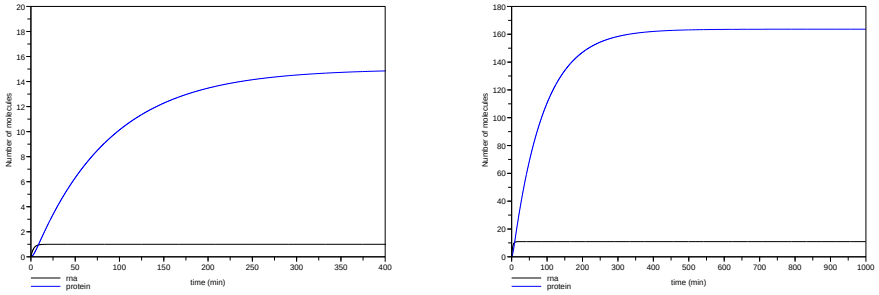


Fig. 3. Constitutive and positive expression using ODE model.

simple semantics and a tractable algebraic theory. Starting with atomic actions and simpler processes, complex processes can be then constructed. The process expressions are defined by guarded processes, parallel composition $P|Q$, nondeterministic choice $P + Q$, replication $!P$, and a restriction operator $(\nu x)P$ creating a local fresh channel x for a process P. Different variants have been used to model molecular interactions [28]. A π-calculus specification of our system is provided by Figure 4. As usual for this type of modelling approach, each chemical element will be represented as a process and its definition will refer to all possible interactions of it. The initial process may be any of $S_{0,i}$. The process called *gene* defines all possible interactions of a constitutive reaction, producing messenger RNA, a positive regulation, leading to a complex denoted by *act-gene*, or negative regulation, that gets the complex *rep-gene*. This process definition corresponds in a P systems model to rules r_1, r_5 and r_8. In this way we can see, at least syntactically, similarities and differences between the two modelling approaches for expressing chemical interactions. More about the use of both P systems and π-calculus to model chemical interactions is provided by [26].

Biochemistry	**π-calculus**
Compartment	Private communication channel
Molecule	Process
Molecular Population	Systems of communicating processes
Biochemical Transformation	Communication channel
Compartment Translocation	Extrusion of a private channel's scope

Table 2. Modelling Principles in π-calculus

Initial processes: $S_{0,1} = gene$; $S_{0,2} = gene \mid act \mid \ldots \mid act$ and $S_{0,3} = gene \mid rep \mid \ldots \mid rep$
Processes:
$gene := \tau_{c_1}.(\, gene \,|\, rna \,) + a_{c_5}?.act\text{-}gene + r_{c_8}?.rep\text{-}gene$
$rna := \tau_{c_2}.(\, rna \,|\, protein \,) + \tau_{c_3}.0$
$protein := \tau_{c_4}.0$
$act := a_{c_5}!.0$
$act\text{-}gene := \tau_{c_6}.(\, act \,|\, gene \,) + \tau_{c_7}.(\, act\text{-}gene \,|\, rna \,)$
$rep := r_{c_8}!.0$
$rep\text{-}gene := \tau_{c_9}.(\, rep \,|\, gene \,)$

Fig. 4. π-calculus model of gene expression.

2.3 Petri nets

Petri nets are a mathematical and computational tool for modelling and analysis of discrete event systems typically with a concurrent behaviour. Petri nets offer a formal way to represent the structure of a discrete event system, simulate its behaviour, and prove certain properties of the system. Petri nets have applications in many fields of system engineering and computer science. Here we only recall some basic concepts of Petri nets and refer to the current literature [9, 23, 24, 25] for details regarding the theory and applications of Petri nets. In particular, we focus only on a specific class of Petri nets called place-transition nets or PT-nets, for short.

Informally, a PT-net is a directed graph formed by two kinds of nodes called places and transitions respectively. Directed edges, called arcs, connect places to transitions, and transitions to places; each arc has associated a weight. Thus, for each transition, one identifies a set of input places, the places which have at least one arc directed to that transition, and a set of output places, the places which the outgoing arcs of that transitions are directed to. Then, a non-negative integer number of tokens is assigned to each place; these numbers of tokens define the state of the PT-net also called the marking of the PT-net. In a PT-net, a transition is enabled when the number of tokens in each input place is greater than or equal to the weight of the arc connecting that place to the transition. An enabled transition can fire by consuming tokens from its input places and producing tokens in its output places; the number of tokens produced and consumed are determined by the weights of the arcs involved. The firing of a transition can be understood as the movement of tokens from some input places to some output places.

More precisely, we give the following definition.

Definition 2. *A* PT-net *is a construct* $N = (P, T, W, M_0)$ *where:* P *is a finite set of* places*,* T *is a finite set of* transitions*, with* $P \cap T = \emptyset$*,* $W : (P \times T) \cup (T \times P) \to \mathbf{N}$ *is the* weight function*,* M_0 *is a multiset over* P *called the* initial marking*, and* L *is a* location mapping*.*

PT-nets are usually represented by diagrams where places are drawn as circles, transitions are drawn as squares, and an arc (x, y) is added between x and y if $W(x, y) \geq 1$. These arcs are then annotated with their weight if this is 2 or more.

Given a PT-net N, the *pre-* and *post-multiset* of a transition t are respectively the multiset $pre_N(t)$ and the multiset $post_N(t)$ such that, for all $p \in P$, $|p|_{pre_N(t)} = W(p, t)$ and $|p|_{post_N(t)} = W(t, p)$. A configuration of N, which is called a *marking*, is any multiset over P; in particular, for every $p \in P$, $|p|_M$ represents the number of *tokens* present inside place p. A transition t is *enabled* at a marking M if the multiset $pre_N(t)$ is contained in the multiset M. An enabled transition t at marking M can *fire* and produce a new marking M' such that $M' = M - pre_N(t) + post_N(t)$ (i.e., for every place $p \in P$, the firing transition t consumes $|p|_{pre_N(t)}$ tokens and produces $|p|_{post_N(t)}$ tokens).

In order to reason about some basic properties, it is convenient to introduce a matrix-based representation for PT-nets. Specifically, let $N = (P, T, W, M_0)$ be a PT-net and let $\pi : P \to |P|$ and $\tau : T \to |T|$ be two bijective functions. We call place j the place p with $\pi(p) = j$, and we call transition i the transition t with $\tau(t) = i$. Then, a marking M is represented as a $|P|$ vector which contains in each position j the number of tokens currently present inside place j. The *incidence matrix* of N is the $|T| \times |P|$ matrix A such that, for every element a_{ij} of A, $a_{ij} = |\pi^{-1}(j)|_{post_N(\tau^{-1}(i))}| - |\pi^{-1}(j)|_{pre_N(\tau^{-1}(i))}|$ (i.e., a_{ij} denotes the change in the number of tokens in place j due to the firing of transition i). A *control vector* u is a $|T|$ vector containing 1 in position i to denote the firing of transition i, 0 otherwise. Thus, if a particular marking M_n is reached from the initial marking M_0 through a firing sequence $u_1, u_2, \ldots, u_n$ of enabled transitions, we obtain

$$M_n = M_0 + A^T \cdot \sum_{k=1}^{n} u_k$$

which represents the *reachable-marking equation*.

The aforementioned representation of a PT-net N allows us to introduce the notions of *P-invariants* and *T-invariants*. P-invariants are the positive solutions of the equation $A \cdot y = 0$; the non-zero entries of a solution y represents the set of places whose total number of tokens does not change with any firing sequence from M_0. T-invariants instead are the positive solutions of the equation $A^T \cdot x = 0$; a solution vector x represents the set of transitions which have to fire from some marking M to return to the same marking M. Then, a PT-net is said to be *bounded* if there exists a $|P|$ vector B such that, for all marking M reachable from M_0, we have $M \leq B$; a PT-net is said to be *alive* if, for all marking M reachable from M_0, there exists at least one transition enabled at marking M.

As pointed out in [10, 22], a PT-net model for a system of molecular interactions can be obtained by representing each molecular species as a different place and each biochemical transformation as a different transition. Tokens inside a place can then be used to indicate the presence of a molecule in certain proportions. This modelling approach is summarised in Table 3. Thus, a biochemical system is represented as a discrete event system whose structural properties are useful for

Biochemistry	PT-net
Molecule	Place
Molecular Population	Marking
Biochemical Transformation	Transition
Reactant	Input Place
Product	Output Place

Table 3. Modelling Principles in PT-nets.

drawing conclusions about the behaviour and structure of the original biochemical system [22]. For instance, P-invariants determine the set of molecules whose total net concentrantions remain unchanged during the application of certain biochemical transformations; T-invariants instead indicate the presence of cyclic reactions which lead to a condition where some reactions are in a state of continuous operation. Also, the property of liveness is useful to determine the absence of metabolic blocks which may hinder the progress of the biochemical system.

Finally, we recall that it was shown in [4, 15, 16] how to transform a P system into a corresponding PT-net. This is done by considering a transition for each rule in the P system that has the left-hand side of the rule as pre-multiset and the right-hand side of the rule as post-multiset. In particular, in order to model the localisation of rules and objects inside the membranes, one considers in the corresponding PT-net a distinct place for each object possibly present inside a membrane. Thus, the transformation of objects inside the membranes and the communication of objects between membranes is mapped into the movement of tokens between places of a PT-net. This translation is briefly illustrated in Table 4. Thus, we have a direct way for obtaining a PT-net representation of a given P

P System	PT-net
Object a inside membrane i	Place a_i
Multiplicity of an object	Number of tokens inside a place
Rule	Transition
Left-hand side of a rule	Input places
Right-hand side of a rule	Output places

Table 4. Translation of a P system into a PT-net.

system model that offers us the possibility of analysing the P system model in terms of PT-net properties. We illustrate this approach by showing in Figure 5 the PT-net translation of the P system model of Figure 1.

Clearly, we have that transition r_i corresponds to rule ri, $1 \leq i \leq 9$. For the PT-net of Figure 5, if we set M_0 as intial marking, where M_0 contains one token in the place *gene*, then we have only consitutive expression; if we set M_0 as having one token in *gene* and n in *act* with $n \geq 1$, then we have positive regulation; if we

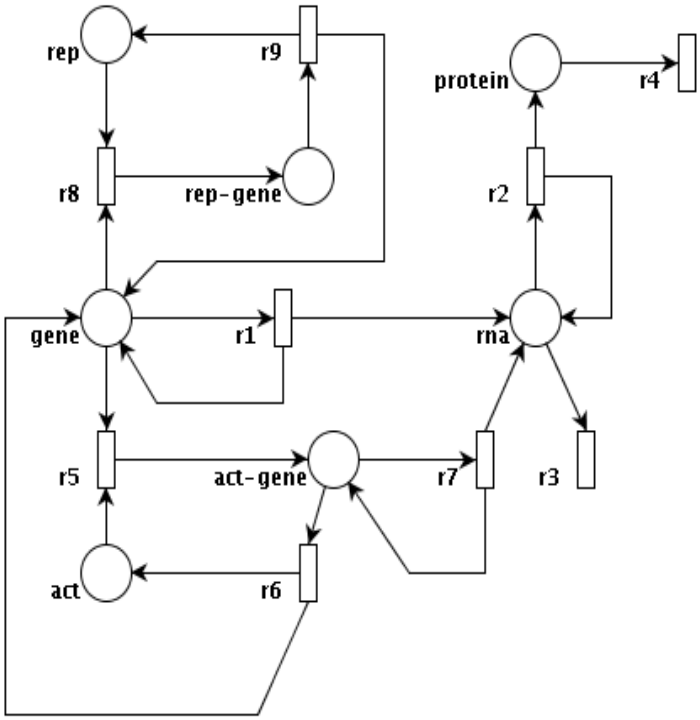


Fig. 5. PT-net representation of gene positive and negative regulation.

set M_0 with one token in *gene* and m in *rep* with $m \geq 1$, then we have negative regulation. The incidence matrix for PT-net of Figure 5 is reported in Appendix 1 together with its P-invariants and T-invariants. The relevance of these invariants with respect to this specifc case study is discussed in Section 5. These are obtained by using PIPE [29], a freely available Petri net tool. As well as this, PIPE allows us to check for the properties of boundedness and liveness (i.e, absence of deadlock).

2.4 Daikon tool

Daikon [5] is a tool that was initially developed to reverse-engineer specifications from software systems. The specifications are in terms of invariants, which are rules that must hold true at particular points as the program executes. To detect invariants the program is executed multiple times and the values of the variables are recorded at specific points (e.g., the start and end of a program function). Daikon infers the invariants by attempting to fit sets of predefined rules to the values of program variables at every recorded program point. Usually the most valuable invariants are preconditions, postconditions and system invariants. These specify

the conditions that must hold between variables before a function is executed, after a function has finished executing, and throughout the program execution respectively. As a trivial example, a precondition for the function $div(a, b)$ that divides a by b would be $b \geq 0$. Daikon provides about 70 predefined invariants [5], such as $x > y$, $a < x < b$, $y = ax + b$, and can also be extended to check for new user-supplied invariants.

The idea of using a set of executions to infer rules that govern system behaviour, as espoused by Daikon, is particularly useful in the context of biological models. The ability to automatically infer invariants from model simulations is useful for the following reasons: (1) obvious invariants will confirm that the model is behaving as it should, (2) anomalous invariants can indicate a fault in the model and its parameter values or (3) could even suggest novel, latent relationships between model variables.

3 Finding functional relationships in raw (wet real) data: Data analysis using Daikon

This section demonstrates the use of Daikon to discover relationships between variables in the output from P system simulations of the gene regulation model. The aim is to identify invariants that govern model behaviour for negative, constitutive and positive gene regulation. Here we select a sample of the generated invariants and show how they relate to the high-level functionality of the system, and how they can be of use for further model analysis. Invariants are only useful if they are representative of a broad range of model behaviour. A single simulation can usually not be considered to be representative, especially if the model is non-trivial and contains stochastic behaviour. For this analysis the model was simulated 30 times, ten times for negative, constitutive and positive regulation respectively.

Using Daikon to generate invariants from simulation output is relatively straightforward. It takes as input two files, one of which declares the types of invariants that are of interest, along with the set of relevant variables for each type of invariant. The other file contains the variable values from model simulations, and lists them under their respective declared invariants. In our case the output is in the form of a linear time series, where variable values are provided for $t = 0...n$ time points in the simulation. To analyse this with Daikon, the declaration file contains the three invariant types described above (preconditions, postconditions and system invariants), along with the key model variables under each type ($gene, act\text{-}gene, rep\text{-}gene, rna, prot, rep, act$). The data trace file maps any variable values at $t = 0$ to the preconditions, the relationship between every pair of variables t and $t-1$ to the postconditions, and the variable values for every value of t to the system invariants. To guarantee accurate invariants, the data trace has to be constructed from a set of simulations that can be deemed to be sufficiently representative of the system's behaviour.

Figure 6 contains a sample of the invariants that were discovered. These provide a number of insights into the behaviour of the system that would be difficult to

	Positive	Negative	Constitutive
Pre-conditions	$gene = 1$ $rna = prot$ $= rep\text{-}gene$ $= rep$ $= act\text{-}gene$ $= 0$ $act = 10$	$gene = 1$ $rna = prot$ $= rep\text{-}gene$ $= act$ $= act\text{-}gene$ $= 0$ $rep = 10$	$gene = 1$ $rna = prot$ $= rep\text{-}gene$ $= rep$ $= act\text{-}gene$ $= act$ $= 0$
Post-condtions	$(prot = 0) \rightarrow (orig(prot) = 0)$ $(rna = 0) \rightarrow (gene = 0)$ $(orig(rna) = 0) \rightarrow (gene = 0)$ $gene \leq rna$ $(prot = 0) \rightarrow (gene = 0)$ $(rna = 0) \rightarrow (orig(act\text{-}gene = 0)$	$act = orig(act)$ $= orig(act\text{-}gene)$ $rna < orig(rep)$ $rep > orig(prot)$	$(prot = 0) \rightarrow (orig(prot) = 0)$ $gene = orig(gene)$ $rep = orig(rep)$
Invariants	$gene = one\ of\ \{0, 1\}$ $rep = rep\text{-}gene$ $= 0$ $0 \leq rna \leq 24$ $0 \leq prot \leq 205$ $act = one\ of\ \{9, 10\}$ $act\text{-}gene = one\ of\ \{0, 1\}$ $(gene \wedge act\text{-}gene) = 0$ $(rna = 0) \rightarrow (prot = 0)$	$gene = one\ of\ \{0, 1\}$ $act = act\text{-}gene$ $= 0$ $rna = one\ of\ \{0, 1\}$ $rep = one\ of\ \{9, 10\}$ $rep\text{-}gene = one\ of\ \{0, 1\}$ $prot = one\ of\ \{0, 1, 2, 3\}$ $(gene \wedge rep\text{-}gene) = 0$ $rna < rep$ $rep > prot$	$rep = rep\text{-}gene$ $= act$ $= act\text{-}gene$ $= 0$ $gene = 1$ $0 \leq rna \leq 7$ $0 \leq prot \leq 32$ $rna \geq rep$

Fig. 6. Invariants discovered by Daikon

ascertain from passively observing the simulations. Here we provide an overview of some of these results.

The preconditions show precisely which model parameters are altered for the positive, negative and constitutive sets of simulations. For all simulations, *gene* starts off as active, and all other variables are zero, apart from the activators variable *act* for positive and the repressors variable *rep* for negative regulation.

The postconditions provide a number of insights into the dynamics of the model because they summarise the rules that govern the change in variable values for every single time-step. For positive regulation we learn that the number of proteins will never decrease back to zero throughout the simulation (protein can only be zero if it was already zero at the previous value of t), and that the gene must become active to produce *rna*, which can only happen when *act-gene* becomes 1. For negative regulation it shows that the amount of activators remain constantly zero. For constitutive regulation, similarly to positive regulation, the number of proteins can never decrease to zero.

The invariants are rules that hold throughout the entire simulation. These usually cover the range of values a variable can hold, e.g., *gene* can either be on or off for positive or negative regulation, but is constantly on for constitutive regulation, or the number of proteins is always between zero and 205 for positive

regulation. It also points out rules that can sometimes be fundamental to the behaviour of the model. For example, in positive regulation *act-gene* and *gene* can never be on at the same time, which makes sense because *act-gene* is responsible for activating the *gene* when it is not active. The same holds for *rep-gene* and *gene* in negative regulation. In positive regulation it also points out that, without any *rna*, there can be no proteins.

These rules provide a number of useful insights into the behaviour of the model, many of which are expected, but some of which may either be anomalous or might identify relationships that had not been previously considered. As an example, the preconditions, which simply summarise the input parameters, are obviously expected, but in practice identified that a small number of our experimental simulations had been mistakenly executed with the wrong parameter values (the precondition for positive regulations stated $act = one\,of\,\{9, 10, 19, 20\}$ instead of just 9 and 10). Rules such as $rna \geq rep$ in constitutive regulation and $rep > prot$ in positive regulation are obviously statistically justified by the simulations, but had not been considered explicitly. New rules like these are useful seeds for further experimentation and analysis, and the following section will show how we have investigated these novel properties with the PRISM model checker.

4 PRISM analysis of the system

Most research in systems biology focuses on the development of models of different biological systems in order to be able to simulate them, accurately enough such as to be able to reveal new properties that can be difficult or impossible to discover through direct experiments. One key question is what one can do with a model, other than just simulate trajectories. This question has been considered in detail for deterministic models, but less for stochastic models. Stochastic systems defy conventional intuition and consequently are harder to conceive. The field is widely open for theoretical advances that help us to reason about systems in greater detail and with finer precision. An attempt in this direction consists of using model checking tools to analyse in an automatic way various properties of the model. Probabilistic model checking is a formal verification technique. It is based on the construction of a precise mathematical model of a system which is to be analysed. Properties of this system are then expressed formally using temporal logic and analysed against the constructed model by a probabilistic model checker. Our current attempt uses a probabilistic symbolic model checking approach based on PRISM (Probabilistic and Symbolic Model Checker) [11, 12]. PRISM supports three different types of probabilistic models, discrete time Markov chains (DTMC), Markov decision processes (MDP) and continuous time Markov chains (CTMC). PRISM supports systems specifications through two temporal logics, PCTL (probabilistic computation tree logic) for DTMC and MDP and CSL (continuous stochastic logic) for CTMC.

In order to construct and analyse a model with PRISM, it must be specified in the PRISM language, a simple, high level, state-based language. The fundamental

components of the PRISM language are modules, variables and commands. A model is composed of a number of modules which can interact with each other. A module contains a number of local variables and commands.

The values of these variables at any given time constitute the states of the module. The space of reachable states is computed using the range of each variable and its initial value. The global state of the whole model is determined by the local state of all modules.

The behaviour of each module is described by a set of commands. A command takes the form:

$$\texttt{[action]}\ \mathtt{g} \rightarrow \lambda_{\mathbf{1}} : \mathbf{u_1} + \cdots + \lambda_{\mathbf{n}} : \mathbf{u_n};$$

The guard $\mathtt{g}$ is a predicate over all the variables of the model. Each update $\mathbf{u_i}$ describes the new values of the variables in the module specifying a transition of the module. The expressions $\lambda_{\mathbf{i}}$ are used to assign probabilistic information, rates, to transitions.

The label `action` placed inside the square brackets are used to synchronise the application of different commands in different modules. This forces two or more modules to make transitions simultaneously. The rate of this transition is equal to the product of the individual rates, since the processes are assumed to be independent.

The main components of a P system are a membrane structure consisting of a number of membranes that can interact with each other, an alphabet of objects and a set of rules associated to each membrane. These components can easily be mapped into the components of the PRISM language using modules to represent membranes, variables to describe the alphabet and commands to specify the rules.

A PRISM specification of our system is provided in Appendix 2. Appendix 3 shows the probability that some molecules concentrations will reach certain values at steady state. The ranges of values provided by Daikon represent an indication of possible levels for various molecular concentrations, but in order to know the likely values around steady states, PRISM provides a set of properties that help in this respect. For example, for positive regulation, Daikon provides the range 0 to 24, for *rna* molecules, but PRISM shows that values between 0 and 15 are more likely to be obtained than values greater than 15, and values over 20 are very unlikely to be reached. These values are also confirmed by the graphs provided by differential equations and P system simulator.

Other properties, suggested by Daikon analysis, like $rna < rep$, $prot < rep$, are also validated by PRISM by showing they take place with a higher probability for values of *rep* less than 5 - see Appendix 4. The average or expected behaviour of the stochastic system is also provided and this is very close to ODE behaviour.

5 Petri net analysis of the system

In this section we will show how different invariants will emerge from the analysis of the Petri net and how Daikon hypotheses are formally verified or new problems are formulated.

T-invariants in Appendix 1 show that:

- If we fire transition $r1$ and then transition $r3$, the current marking of the newtork remains unchanged because we first produce a molecule of *rna* and then we consume it; the same happens if we first fire transition $r7$ and than transition $r3$, or if we first fire transition $r2$ and then transition $r4$ (i.e., we first produce a molecule of *protein* and then we consume it);
- The operation of binding the activator to the gene and its de-binding are one the reverse of the other, hence firing transition $r5$ followed by transition $r6$ (or vice versa) has no effect on the current marking; these two transitions consitute a continuous loop;
- The operation of binding the repressor to the gene and its de-binding are one the reverse of the other, hence firing transition $r8$ followed by transition $r9$ (or vice versa) has no effect on the current marking; these two transitions consitute a continuous loop.

P-invariants computed by PIPE, in Appendix 1, for some initial marking with one element in *gene*, n in *act* and m in *rep*, where $n, m \geq 0$ show that:

- the gene is always present and it can assume three different states: *gene*, *act-gene*, and *rep-gene*; these three states are mutually exclusive; in the case of constitutive expression (i.e, $n = m = 0$), we have $M(gene) = 1$ indicating that the gene is always present - this confirms Daikon invariant $gene = 1$; in the case of positive regulation (i.e., $n \geq 1$ and $m = 0$), we have $M(gene) + M(act\text{-}gene) = 1$ indicating two mutually exclusive states - this confirms Daikon invariant $gene \wedge act\text{-}gene = 0$; in the case of negative regulation (i.e., $m \geq 1$ and $n = 0$, we have $M(gene) + M(rep\text{-}gene) = 1$ indicating two mutually exclusive states - this confirms Daikon invariant $gene \wedge rep\text{-}gene = 0$;
- for positive regulation, the number of activator molecules cannot be increased but can be decreased only by 1 - similar invariant is found by Daikon;
- for negative regulation, the number of repressor molecules cannot be increased but can be decreased only by 1 - similar invariant is found by Daikon.

PIPE also shows that the network is not bounded but it is alive. In fact, *gene* is always present and we can keep firing transition $r1$ to increase indefinitely the amount of *rna*. The liveness here comes from the above invariant and shows that the system will be working forever. The boundedness instead produces a result that apparently contradicts PRISM findings, where the probability that the number of *rna*'s is greater than 7 is almost 0! This comes from the fact that PIPE uses a non-deterministic system instead of a probabilistic one considered by PRISM and P system simulator. It will be interesting to check this property with a probabilistic Petri net tool.

6 Conclusions

In this paper we have investigated the concerted use of different methods, and shown how these can provide complementary insights into different facets of biological system behaviour. Individual modelling techniques have their own respective benefits and usually excel at reasoning about a system from a particular perspective. This paper shows how these benefits can be leveraged by using different modelling techniques in concert.

As a case study, we have constructed a P system model of a small gene expression system and produced equivalent specifications using Petri net and π-calculus approaches. Simulations of the P system model were analysed by Daikon (to identify potential rules that govern model output), and some of the most interesting suggested rules were checked using the PRISM probabalistic model checker. The Petri net model was analysed with PIPE, a general Petri net analysis tool. The results show how analysis results from different models of the same system are useful for the purposes of both validating and improving each other.

The gene expression model was chosen because it is manageable, and thus forms a useful basis for a case study to compare different modelling techniques. Our future work will apply the techniques shown in this paper to a larger and more realistic case study. This should provide further insights into the benefits that arise in modelling increasingly complex systems when the modeller is increasingly reliant upon the use of various automated tools to study the model behaviour.

Acknowledgements

The research of Francesco Bernardini is supported by NWO, Organization for Scientific Research of The Netherlands, project 635.100.006 "VIEWS". The authors would like to thank the anonymous reviewers for their comments and suggestions that have helped to improve the paper.

References

1. Bianco, L., Fontana, F. (2006) Towards a Hybrid Metabolic Algorithm. In Hoogeboom, H.J., Păun, Gh. Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing, Seventh International Workshop, WMC7, Leiden, The Netherlands*, volume 4361 of *Lecture Notes in Computer Science*, pages 183–196. Springer Verlag.
2. Blundell, M., Kennell, D. (1974) Evidence for Endonucleolytic Attack in Decay of Lac Messenger RNA in Escherichia Coli, *J. Mol. Biol.*, **83** 143 – 161.
3. Calder, M., Vyshemirsky, V., Gilbert, D., Orton, R. (2006) Analysis of Signalling Pathways Using Continuous Time Markov Chains, *Transactions on Computational Systems Biology VI, LNBI* **4220** 44 – 67.
4. Dal Zilio, S., Formenti, E. (2004) On the Dynamics of PB Systems: A Petri Net View. In Martín-Vide, C., Mauri, G., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing, Fourth International Workshop, WMC4, Tarragona, Spain*, volume 2933 of *Lecture Notes in Computer Science*, pages 153–167. Springer Verlag.

5. Ernst, M., Cockrell, J., Griswold, W., Notkin, D. (2001) Dynamically Discovering Likely Program Invariants to Support Program Evolution, *IEEE Transactions on Software Engineering*, **27** (2) 99–123.
6. Fontana, F., Bianco, L., Manca, V. (2005) P Systems and the Modelling of Biochemical Oscillations. In Freund, R., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing, Sixth International Workshop, WMC6, Vienna, Austria*, volume 3850 of *Lecture Notes in Computer Science*, pages 199 – 208. Springer Verlag.
7. Fontana, F., Manca, V. (2007) Discrete Solutions of Differential Equations by Metabolic P Systems, *Theoretical Computer Science*, **372**(2-3) 165 – 182.
8. Gillespie, D.T. (1977) Exact Stochastic Simulation of Coupled Chemical Reactions, *The Journal of Physical Chemistry*, **81**(25) 2340–2361.
9. Girault, C., Valk., R. (2003) *Petri Nets for Systems Engineering*. Springer Verlag.
10. Goss, P.J.E., Peccoud, J. (1998) Quantitative Modeling of Stochastic Systems in Molecular Biology using Stochastic Petri Nets., *Proc. Natl. Acad. Sci. USA*, **95** 6750–6755.
11. Heath, J., Kwiatkowska, M., Norman, G., Parker, D., Tymchyshyn. (2006) Probabilistic Model Checking of Complex Biological Pathways, In *Proc. CMSB'06,* LNCS, to appear.
12. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D. (2006) PRISM: A Tool for Automatic Verification of Probabilistic Systems, volume 3920 of *Lecture Notes in Computer Science*, pages 441 – 444. Springer Verlag.
13. Hlavacek, S., Savageau, M.A. (1995) Subunit Structure of Regulator Proteins Influences the Design of Gene Circuitry Analysis of Perfectly Coupled and Uncoupled Circuits, *J. Mol. Biol.*, **248** 739 – 755.
14. Kennell, D., Riezman, H. (1977) Transcription and Translation Initiation Frequencies of the Escherichia Coli Lac Operon, *J. Mol. Biol.*, **114** 1–21.
15. Kleijn, K., Koutny, K. (2007) Synchrony and Asynchrony in Membrane Systems. In Hoogeboom, H.J., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing, Seventh International Workshop, WMC7, Leiden, The Netherlands*, volume 4361 of *Lecture Notes in Computer Science*, pages 66–85. Springer Verlag.
16. Kleijn, K., Koutny, M., Rozenberg, G. (2006) Towards a Petri Net Semantics for Membrane Systems. In Freund, R., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) *Membrane Computing, Sixth International Workshop, WMC6, Vienna, Austria*, volume 3850 of *Lecture Notes in Computer Science*, pages 292–309. Springer Verlag.
17. Milner, R. (1999) *Communication and Mobile Systems: The π-calculus*. Cambridge University Press.
18. Păun, Gh. (2000) Computing with Membranes, *Journal of Computer and System Sciences*, **61**(1) 108 – 143.
19. Pérez-Jiménez, M.J., Romero-Campero, F.J. (2006) P Systems, a New Computationl Modelling Tool for Systems Biology, *Transactions on Computational Systems Biology VI, LNBI* **4220** 176–197.
20. Pérez-Jiménez, M.J., Romero-Campero, F.J. Modelling Gene Expression Control Using P Systems: The Lac Operon, A Case Study, to appear.
21. Pescini, D., Besozzi, D., Mauri, C., Zandron, C. (2006) Dynamical Probabilistic P systems, *International Journal of Foundations of Computer Science* **17**(1) 183 – 204.
22. Reddy, V.N., Liebman, M.N., Mavrouniotis, M.L. (1996) Qualitative Analysis of Biochemical Reaction Systems, *Computers in Biology & Medicine*, **26**(1) 9–24.
23. Reisig, W. (1998) *Elements of Distributed Algorithms. Modelling and Analysis with Petri Nets*. Springer Verlag.

24. Reisig, W., Rozenberg, G. (eds.) (1998) *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer Verlag.
25. Reisig, W., Rozenberg, G. (eds.) (1998) *Lectures on Petri Nets II: Applications*, volume 1492 of *Lecture Notes in Computer Science*. Springer Verlag.
26. Romero-Campero, F.J., Gheorghe, M., Ciobanu, G., Auld, J.M., Pérez-Jiménez, M.J. (2007) Cellular Modelling Using P Systems and Process Algebra, *Progress in Natural Science*, **17**(4) 375–383.
27. Romero-Campero, F.J., Pérez-Jiménez, M.J. A Model of the Quorum Sensing System in Vibrio Fischeri Using P Systems, submitted.
28. Regev, A., Shapiro, E. (2004) The π-calculus as an Abstraction for Biomolecular Systems. In Ciobanu, G., Rozenberg, G. (eds.) *Modelling in Molecular Biology*. Springer Verlag.
29. Platform Independent Petri Net Editor:
`http://pipe2.sourceforge.net`
30. P System Simulator:
`http://www.dcs.shef.ac.uk/ marian/PSimulatorWeb/PSystemMF.htm`

APPENDIX 1

Transpose of the incidence matrix for PT-net of Figure 5:

```
          r1 r2 r4 r3 r5 r7 r6 r8 r9
gene       0  0  0  0 -1  0  1 -1  1
rna        1  0  0 -1  0  1  0  0  0
protein    0  1 -1  0  0  0  0  0  0
act        0  0  0  0 -1  0  1  0  0
act-gene   0  0  0  0  1  0 -1  0  0
rep        0  0  0  0  0  0  0 -1  1
rep-gene   0  0  0  0  0  0  0  1 -1
```

which shows the variations on the number of tokens determined by each transition.

T-invariants obtained in PIPE:

```
r1     1      0      0     0     0
r2     0      1      0     0     0
r4     0      1      0     0     0
r3     1      0      0     1     0
r5     0      0      1     0     0
r7     0      0      0     1     0
r6     0      0      1     0     0
r8     0      0      0     0     1
r9     0      0      0     0     1
```

P-invariants computed by PIPE for some initial marking contains one token in *gene* n in *act* and m in *rep*, with $n, m \geq 0$:

```
gene     1     0     0
rna      0     0     0    M(gene) + M(act-gene) + M(rep-gene) = 1
protein  0     0     0    M(act) + M(act-gene) = n
act      0     1     0    M(rep) + M(rep-gene) = m
act-gene 1     1     0
rep      0     0     1
rep-gene 1     0     1
```

APPENDIX 3

Ranges of molecules

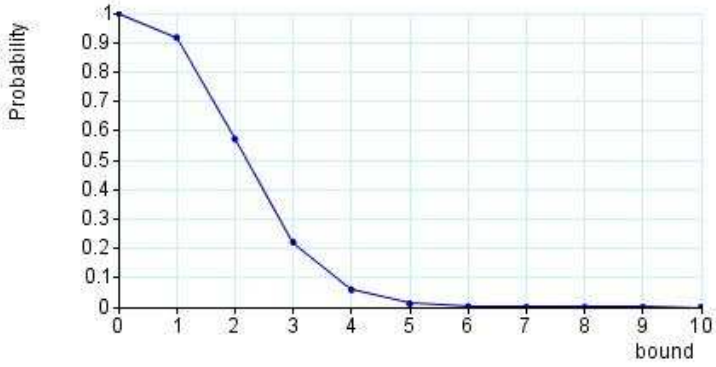
```
P = ? [ true U <= T rna > bound ]

P = ? [ true U <= T protein > bound ]
```

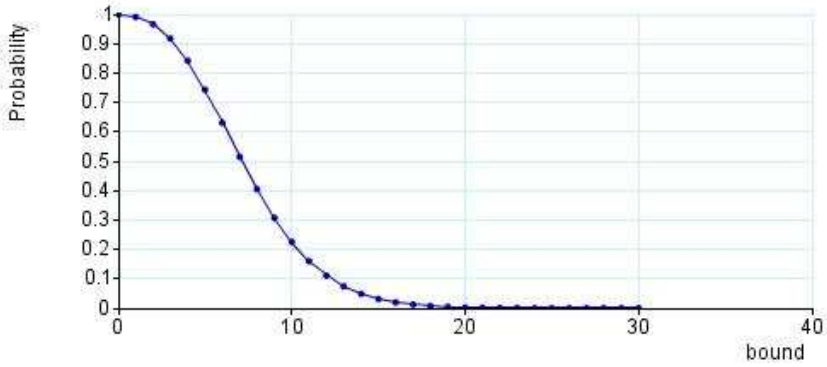
APPENDIX 2

```
// Gene expression control
// Model is stochastic
stochastic
// Bounds to the number of molecules
const int rna_bound;
const int protein_bound;
const int number_activators;
const int number_repressors;
const int initact;
const int initrep;
// Stochastics constants associated with each command/rule/molecular interaction
const double c1 = 0.347;        // [ gene ]_b -c1-> [ gene + rna ]_b
const double c2 = 0.174;        // [ rna ]_b -c2-> [ rna + protein ]_b \\
const double c3 = 0.347;        // [ rna ]_b -c3-> [   ]_b
const double c4 = 0.0116;       // [ protein ]_b -c4-> [   ]_b
const double c5 = 6.6412087;    // [ act + gene ]_b -c5-> [ actgene ]_b
const double c6 = 0.6;          // [ actgene ]_b -c6-> [ act + gene ]_b
const double c7 = 3.47;         // [ actgene ]_b -c7-> [ actgene + rna ]_b
const double c8 = 6.6412087;    // [ rep + gene ]_b -c8-> [ repgene ]_b
const double c9 = 0.6;         // [ repgene ]\_b -c9-> [ rep + gene ]_b
// Module representing a bacterium
module bacterium
    gene : [ 0 .. 1 ] init 1;
    actgene : [ 0 .. 1 ] init 0;
    repgene : [ 0 .. 1 ] init 0;
    act : [ 0 .. 1 ] init initact;
    rep : [ 0 .. 1 ] init initrep;
    rna : [ 0 .. rna_bound ] init 0;
    protein : [ 0 .. protein_bound ] init 0;
    // [ gene ]_b -c1-> [ gene + rna ]_b
    [ ] gene = 1 & rna < rna_bound -> c1 : (rna' = rna + 1);
    // [ rna ]_b -c2-> [ rna + protein ]_b
    [ ] rna > 0 & protein < protein_bound -> c2*rna : (protein' = protein + 1);
    // [ rna ]_b -c3-> [ ]_b
    [ ] rna > 0 -> c3*rna : (rna' = rna - 1);
    // [ protein ]_b -c4-> [   ]_b
    [ ] protein > 0 -> c4*protein : (protein' = protein - 1);
    // [ act + gene ]_b -c5-> [ actgene ]_b
    [ ] act = 1 & gene = 1 -> c5*number_activators : (gene' = 0) & (act' = 0) & (actgene' = 1);
    // [ actgene ]_b -c6-> [ act + gene ]_b
    [ ] actgene = 1 & act = 0 -> c6 : (actgene' = 0) & (act' = 1) & (gene' = 1);
    // [ actgene ]_b -c7-> [ actgene + rna ]_b
    [ ] actgene = 1 & rna < rna_bound -> c7 : (rna' = rna + 1);
    // [ rep + gene ]_b -c8-> [ repgene ]_b
    [ ] rep = 1 & gene = 1 -> c8*number_repressors : (gene' = 0) & (rep' = 0) & (repgene' = 1);
    // [ repgene ]_b -c9-> [ rep + gene ]_b
    [ ] repgene = 1 & rep = 0 -> c9 : (repgene' = 0) & (rep' = 1) & (gene' = 1);
endmodule
```

Constitutive regulation

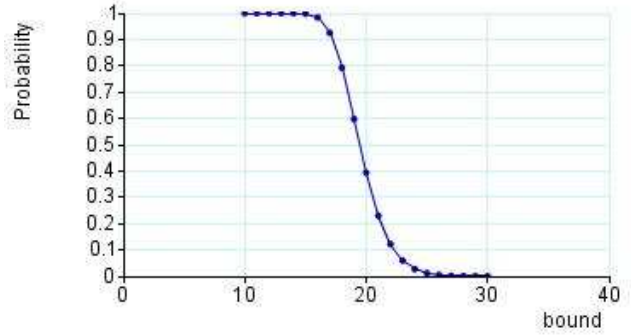
```
rna <= 7

rna >= 0

prot <= 32

prot >= 0
```

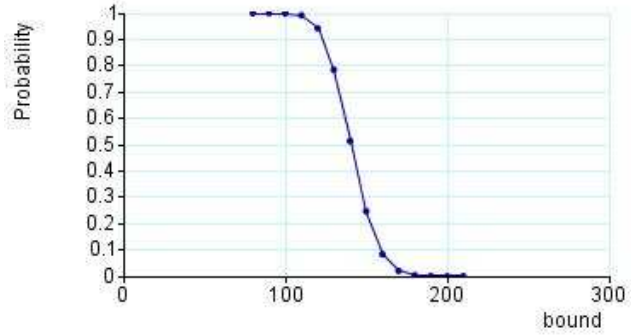




Positive regulation

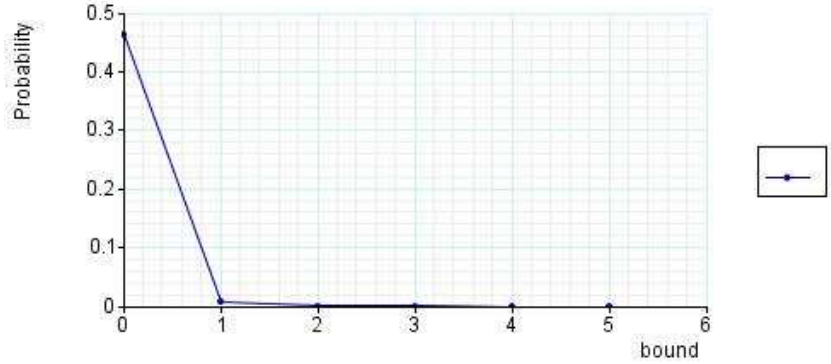
```
rna <= 24

rna >= 0

prot <= 205

prot >= 0
```

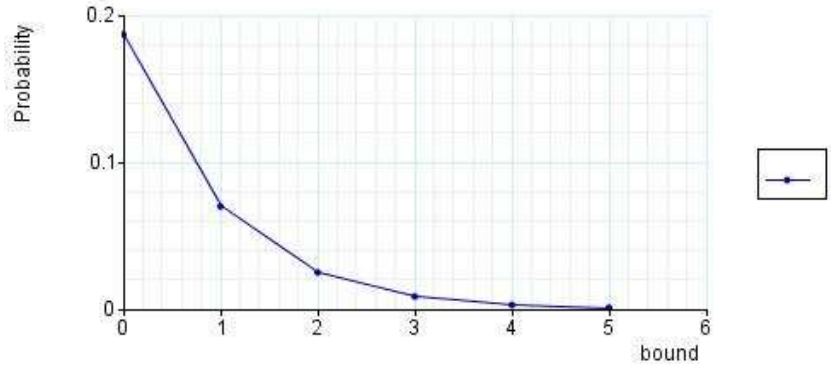




Negative regulation

```
rna one of { 0, 1 }

prot one of { 0, 1, 2, 3 }
```

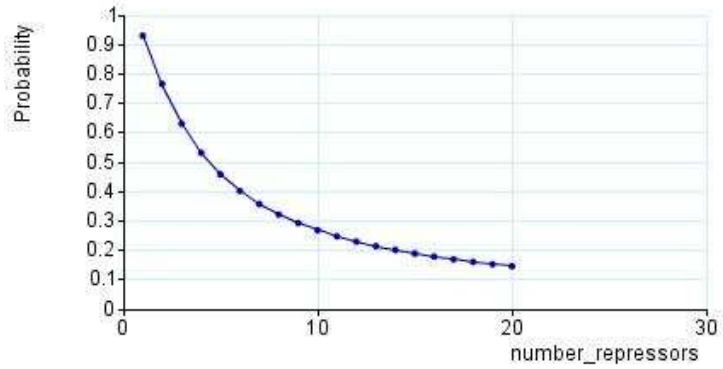


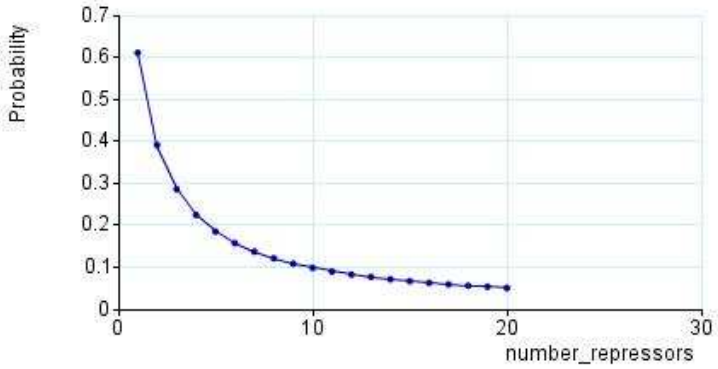


APPENDIX 4

Relationship between the number of repressors and rna and protein molecules.

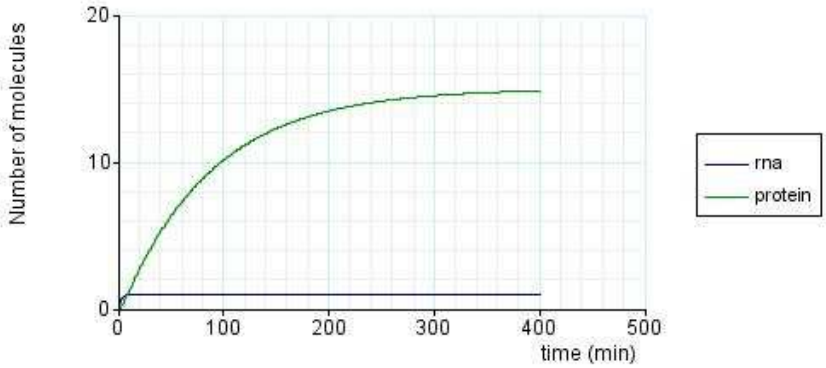
```
rna < rep

rep > prot

P = ? [ true U<=T rna > rep ]

P = ? [ true U<=T protein > rep ]
```

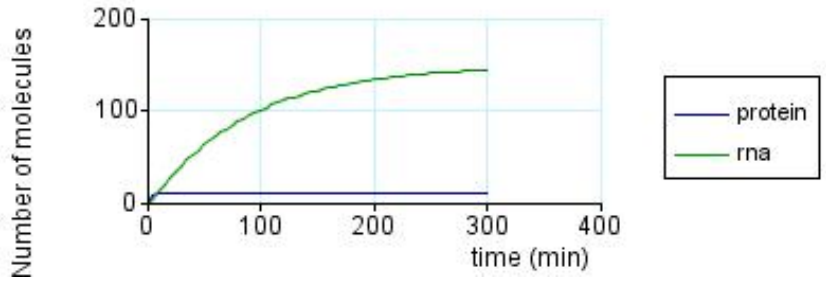


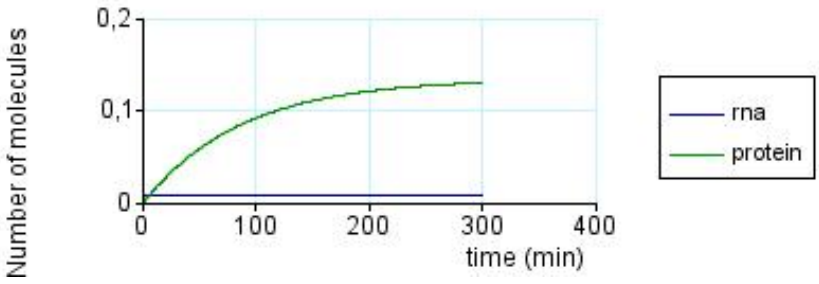


Expected number of molecules

R = ? [I = T]







Other invariants

`P = ? [ true U gene = actgene ]` ⇒ `Result: 0.0`

`P = ? [ true U gene = repgene ]` ⇒ `Result: 0.0`