<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Availability</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Getting Started</td>
<td>5</td>
</tr>
<tr>
<td>2.1</td>
<td>Quick Start</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Tutorials</td>
<td>13</td>
</tr>
<tr>
<td>2.3</td>
<td>Documentation</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>Model Repository</td>
<td>79</td>
</tr>
<tr>
<td>3.1</td>
<td>The Repressilator</td>
<td>79</td>
</tr>
<tr>
<td>3.2</td>
<td>Pulse Generator</td>
<td>86</td>
</tr>
<tr>
<td>3.3</td>
<td>Automatic Discovery of Pulse Generators</td>
<td>96</td>
</tr>
<tr>
<td>3.4</td>
<td>Auxin Transport</td>
<td>98</td>
</tr>
<tr>
<td>4</td>
<td>How To Acknowledge</td>
<td>101</td>
</tr>
<tr>
<td>4.1</td>
<td>Related Publications</td>
<td>101</td>
</tr>
</tbody>
</table>
Infobiotics workbench is a computational framework implementing a synergy between executable biology, multi-compartmental stochastic simulations, formal model analysis and structural/parameter model optimisation for computational systems and synthetic biology. It provides a user-friendly front-end allowing the modeller to design in-silico experiments, analyse and visualise results using its four components:

- **A modelling language** based on P systems which allows modular and parsimonious multi-cellular model development including geometric information.

- **A multi-compartmental stochastic simulator** based on Gillespie’s Stochastic Simulation Algorithm for multi-cellular systems.

- **Formal model analysis** using the stochastic model checkers PRISM and MC2 for the study of temporal and spatial model properties.

- **Structural and parameter model optimisation** using evolutionary algorithms to automatically generate models whose dynamics match specified targets.
Binaries are available for Windows XP, Vista and 7, Mac OS X 10.6 and Linux (deb / rpm). Source code is also available to download under the GNU GPL v3 license.
A quick start, tutorials and the complete documentation are available from the links below:

### 2.1 Quick Start

In this quick start we will walk through an example based on gene negative autoregulation (NAR) to explain the basics of getting started with the Infobiotics Workbench. Alternatively you can follow our video tutorial.

1. First you need to download and install Infobiotics Workbench from this link.
2. Download this example containing the NAR model and unzip it to your favourite location.
3. Open the Infobiotics Workbench by double clicking on the corresponding icon located on your desktop (Windows) or by choosing it from your Applications menu (Mac/Linux). The following window will appear showing the different components: simulation, model checking (PRISM and MC2) and optimisation - without the open model files shown.

1. Click on the Simulation button on the toolbar to open up the dialog window below allowing you to specify your simulation parameters.

1. Load the simulation parameter file simulation.params by selecting Load from the dialog toolbar and navigating to the location of the NAR model.

2. Run your simulations by clicking on the Perfrom button at the bottom of the simulation dialog window.

3. Once your simulations have finished the following tab will appear to allow you to plot the results.

   1. Plot the average number of molecules over time for all species in all compartments by checking All under Runs, Species and Compartments, then clicking on the first button (‘timeseries’) in the bottom right corner.

   2. A preview window will appear that allows you to combine the various timeseries in different ways. Select all the graphs (Ctrl-A) and click the Stack button to view each timeseries with the same time axis but individual molecules axes.

The Infobiotics Workbench is not limited to performing simulations, you can apply other techniques to analyse and manipulate your models. Visit the links below if you are interested in the different components of our workbench.

#### 2.1.1 Analysis of model properties

1. Click on the Model checking (PRISM) button on the toolbar to open up the dialog window below that will allow you to specify the properties to analyse in your model.

1. Load the model checking parameter file model_checking_prism.params by clicking Load from the dialog toolbar and navigating to the location of the negative autoregulation model.
Infobiotics Workbench, Release 0.0.1

Chapter 2. Getting Started

Infobiotics Dashboard 0.0.9

Simulation Model checking (PRISM) Model checking (MC2) Optimisation

library.plb

1 Author: Francisco J. Romaro-Camero #
2 Date: 14 May 2010 #
3 Description: A model of a gene that regulates itself negatively #
4
5 system negativeAutoRegulation
6 # Molecular species in the system #
7 alphabet
gene1
gene2
proteint
proteint_0
rnt
endAlphabet
8
9 system consists of a single compartment called bacterium #
10 compartments
bacterium
endCompartments
11 # The initial number of molecules present in the system #
12 initialMultiset
initialMultiset bacterium
gene1_0
endInitialMultiset
13 # The rules describing the molecular interactions in the different compartments of the system #
14 ruleSets
15
16 ruleSet bacterium
17 # Unregulated expression of gene 1 #
18 UnReg(1,1,(0,05),(0,06),(0,04),(0,0000000056),(bacterium)) from basicLibrary.plb
19 # Negative regulation of gene 1 by the protein product of gene 1 #
20 NegReg((1,1),(0.0056,0.127),(bacterium)) from basicLibrary.plb
21
22 endRuleSet
2.1. Quick Start

Infobiotics Workbench, Release 0.0.1
2. Check the parameters then run the experiment by clicking on the **Perform** button.
3. Once the experiment has finished the following tab will appear automatically showing a plot of the results.

For more details on how to use the **Infobiotics Workbench** you can read our tutorials.

### 2.1.2 Structural and Parameter Optimisation

1. Click on the **Optimisation** button on the toolbar to open up the dialog window below that will allow you to specify how the structure and parameter optimisation is performed.
Chapter 2. Getting Started
1. Load the optimisation parameter file *optimisation.params* by clicking the **Load** button on the toolbar and navigating to the location of the negative autoregulation model.

2. Run the predefined optimisation experiment by clicking on the **Perform** button. This process should take approximately one minute.

3. Once the experiment has finished the following dialog will appear automatically, detailing best model found and plotting its simulated behaviour against the desired target behaviour.

For more details on how to use the **Infobiotics Workbench** you can read our **tutorials**.

**2.2 Tutorials**

The tutorials below will take you through the most important features of **Infobiotics workbench** using running examples to illustrate them. Infobiotics workbench is available for installation for a variety of platforms. Please download and install it from this [link](#) following the instructions provided.

Infobiotics workbench provides a user-friendly front-end allowing the modeller to specify cellular models, analyse and optimise them. Start the Infobiotics workbench by double clicking on the corresponding icon located on your desktop (Windows) or by choosing it from your Applications menu (Mac/Linux). The following window will appear.
The links below will guide you through the different features of our workbench. Video tutorials are available in each tutorial to show you how to use the different components integrated in the workbench.

### 2.2.1 Tutorial Using the Infobiotics Modelling Language

With this tutorial you will learn how to use the **Infobiotics workbench** using its own modelling language for the specification of your models. We will illustrate its different features using an example comparing three basic gene regulatory mechanisms, namely, **gene unregulated expression** (UnReg), **positive autoregulation** (PAR) and **negative autoregulation** (NAR).

The example used in the tutorial can be downloaded from this link: autoregulation model. Please download and extract it to your favourite location.

The links below will guide you through the different features of our workbench. Video tutorials are available in each section to show you how to use the different components.

---

**Model Specification and Building**

The **Infobiotics modelling language** is based on **Stochastic P-systems**, SP-system for short. It allows you to specify your models in a *parsimonious* and *incremental* way. Typically, **multi-cellular models** are specified in our modelling language using *libraries of re-usable modules, models of individual cell types* and *geometric distributions* of clones of these cell types. We will illustrate this using our running example based on **gene autoregulation**.

**Libraries of reusable modules**

Libraries of modules must be specified using text files with the extension *plb*. You can open the file containing the library in our running example, basicLibrary.plb, by selecting **File -> Open text file** from the upper menu bar and navigating to its specific location.

Our libraries are specified using the key workds *libraryOfModules ... endLibraryOfModules*. Each library is identified with a name, basicLibrary in our running example and consists of a collection of modules of molecular interactions. In our case we have three different modules, **UnReg**, **PosReg** and **NegReg** describing constitutive gene expression,
gene positive regulation and gene negative regulation respectively. **Comments** can be included using text enclosed between # symbols. Below you have the library used in the autoregulation example:

```plaintext
# Author: Francisco J. Romero-Campero #
# Date: May 2010 #
# Description: A library containing basic gene regulatory mechanisms #

libraryOfModules basicLibrary

# A module representing the unregulated expression of a gene X #
UnReg({X},{c_1, c_2, c_3, c_4},{l}) =
{
  rules:
  # Transcription of geneX #
  r1: [ geneX ]_l -c_1-> [ geneX + rnaX ]_l
  # Degradation of the RNA #
  r2: [ rnaX ]_l -c_2-> [ ]_l
  # Translation of the RNA #
  r3: [ rnaX ]_l -c_3-> [ rnaX + proteinX ]_l
  # Degradation of the protein #
  r4: [ proteinX ]_l -c_4-> [ ]_l
}

# A module representing the positive regulation of a protein X over a gene Y #
PosReg({X,Y},{c_1,c_2,c_3,c_4,c_5,c_6},{l}) =
{
  rules:
  # Binding and debinding of the transcription factor proteinX to geneY #
  r1: [ proteinX + geneY ]_l -c_1-> [ proteinX_geneY ]_l
  r2: [ proteinX_geneY ]_l -c_2-> [ proteinX + geneY ]_l
  # Transcription of geneY when proteinX is bound to its promoter #
  r3: [ proteinX_geneY ]_l -c_3-> [ proteinX_geneY + rnaY ]_l
  r4: [ rnaY ]_l -c_4-> [ ]_l
  r5: [ rnaY ]_l -c_5-> [ rnaY + proteinY ]_l
  r6: [ proteinY ]_l -c_6-> [ ]_l
}

# A module representing the negative regulation of a protein X over a gene Y #
NegReg({X,Y},{c_1,c_2},{l}) =
{
  rules:
  # Binding and debinding of the transcription factor proteinX to gene Y #
  r1: [ proteinX + geneY ]_l -c_1-> [ proteinX_geneY ]_l
  r2: [ proteinX_geneY ]_l -c_2-> [ proteinX + geneY ]_l
}

endLibraryOfModules
```

A **module** is identified with a name and is associated three sets of variables, i.e. `UnReg({X},{c_1, c_2, c_3, c_4},{l})`). The first set of variables can be instantiated with names of specific molecular species. The second set of variables can be instantiated with numerical values capturing the rates of the molecular interactions represented in the module. Finally, the third set of variables can be instantiated with the names of the compartments involved in the molecular interactions.

A module **encapsulates** a set of molecular interactions represented by rules (or other modules) that may use some of the module variables. For example, rules `r1` and `r2` from the `NegReg` module describe the binding/debinding of `proteinX` to/from `geneY`. These processes take place at rates `c_1` and `c_2` inside compartment `l` represented using square brackets. Note that `X, Y, c_1, c_2` and `l` are module variables that can be instantiated with specific values.
Infobiotics Workbench, Release 0.0.1

r1: [ proteinX + geneY ]_l -c_1-> [ proteinX_geneY ]_l
r2: [ proteinX_geneY ]_l -c_2-> [ proteinX + geneY ]_l

Other attributes can be associated with our modules such as types and specific DNA sequences see this example or the documentation for more details.

Cell types

The Infobiotics modelling language supports the specification of cell types in two different formats. You can specify your cell types using SBML, this example. Alternatively, if you want to use modules of molecular interactions from different libraries defined previously you need to specify your cell type using our modelling language. In this last case, you must use a text file with the extension sps, from SP-system.

Our autoregulation example uses three different cell types carrying the same gene under three different regulatory mechanisms, unregulated expression (UnReg.sps), positive autoregulation (PAR.sps) and negative autoregulation (NAR.sps). Open the model of the last cell type by selecting File -> Open text file from the upper menu bar and choosing the file NAR.sps:

```plaintext
# Author: Francisco J. Romero-Campero #
# Date: July 2010 #
# Description: A model of a cell type carrying a gene that regulates itself negatively #

SPsystem negativeAutoregulation

# Molecular species in the system #
alphabet
genel
protein1
protein1_gene1
rna1
signal1
endAlphabet

# The system consists of two compartments called media and bacterium. The bacterium is embedded in the media #
compartments
media
bacterium inside media
endCompartments

# The initial number of molecules present in the system #
initialMultisets
initialMultiset bacterium
genel 1
endInitialMultiset
endInitialMultisets

# The rules describing the molecular interactions in the different compartments #
# of the system #
ruleSets

    # Molecular interactions involving the compartment bacterium #
    ruleSet bacterium
    # Unregulated expression of gene 1 #
    UnReg({1},{3,0.07,3,0.01},{bacterium}) from basicLibrary.plb
```
# Negative regulation of gene 1 by the protein product of gene 1 #
NegReg({1,1},{1,0.8},{bacterium}) from basicLibrary.plb

# Protein1 is an enzyme that synthesizes signal1 #
r1: [ protein1 ]_bacterium -c1-> [ protein1 + signal1 ]_bacterium c1 = 0.001

# Signal1 diffuses freely outside bacteria #
r2: [ signal1 ]_bacterium -c2-> signal1 [ ]_bacterium c2 = 0.001

# Signal1 can be degraded inside bacteria #
r3: [ signal1 ]_bacterium -c3-> [ ]_bacterium c3 = 0.0001

# Molecular interactions involving the compartment media #
ruleset media

# Signal1 diffuses freely inside bacteria #
r1: signal1 [ ]_bacterium -c1-> [ signal1 ]_bacterium c1 = 0.001

# Signal1 can be degraded in the media #
r2: [ signal1 ]_bacterium -c2-> [ ]_bacterium c2 = 0.0001

# Signal1 diffuses freely to neighbouring media #
r3: [ signal1 ]_bacterium =(1,0)->[ ] -c3-> [ ]_bacterium =(1,0)= [ signal1 ] c3 = 0.0002

A cell type is specified using the key words SPsystem ... endSPsystem. Each cell type is identified with a name, negativeAutoregulation in our example above, and specifies the following components of a single-cell model:

1. The molecular species in the cell type are specified as a list of names (gene1, protein1, etc.) enclosed between the key words alphabet ... endAlphabet.

2. The compartments of the cell type are listed using the key words compartments ... endCompartments. Each compartment is specified using its name. If a compartment is embedded in another one the key word inside is used. For instance, the cell type above consists of two compartments, media and bacterium. The compartment bacterium is contained in the media which is specified as bacterium inside media.

3. The initial number of molecules is specified with the key words initialMultisets ... endInitialMultisets. Each compartment is specifically associated with its initial number of molecules using the key words initialMultiset ... endInitialMultiset and its name. The number of molecules is specified as a list of molecular species names followed by a positive integer number. In our example, only a single copy of gene1 is initially present in the compartment bacterium.

4. The molecular interactions taking place inside or between compartments are enumerated within the key words ruleSets ... endRuleSets. The molecular interactions associated with a compartment with name CompName are specified as a list of rules and instantiated modules enclosed between the key words ruleset ... endRuleSet and identified with CompName. The library file where the modules are defined must be specified after the module instantiation using the key word from. In our example, the interactions involving the compartment bacterium are specified using two instantiated modules defined in the basicLibrary.plb file (UnReg({1,1},{3,0.07,3,0.01},{bacterium}) and NegReg({1,1},{1,0.8},{bacterium})) and three rules describing the synthesis of signal1 by protein1 (rule r1), the diffusion of signal1 outside the bacterium (rule r2) and the degradation of signal1 (rule r3). The values of the stochastic constants associated with these rules are also stated at the end of the rule specifications. Note that the outermost compartment of a cell type can diffuse molecules to neighbouring cells using rules of the same form as rules r3, r4, r5 and r6 in the media compartment. A vector v is associated with this type of rule. The application of a rule of this form in a cell located at position p moves the corresponding molecules to the cell located at position p+v.
Geometric distribution

The **Infobiotics modelling language** allows you to specify the spatial distribution of cells in multi-cellular systems. The characteristic geometry of such systems is captured using **finite point lattices** (a grid of regularly distributed spatial points) that must be specified in files with the extension `lat`. Our running example uses a rectangular lattice similar to the one shown below.

Open the rectangular lattice used in our example by selecting **File -> Open text file** from the upper menu bar and choosing the file `rectangular.lat`:

```plaintext
lattice rectangularLattice
  # Dimension of the lattice and lower/upper bounds #
  dimension 2
  xmin 0
  xmax 39
  ymin 0
  ymax 9

  # Parameters used in the definition of the rest of components defining the lattice #
  parameters
    parameter b1 value = 2
    parameter b2 value = 1
  endParameters

  # Basis vector determining the points in the lattice #
  # in this case we have a rectangular lattice      #
  basis
    (b1,0)
    (0,b2)
  endBasis

  # Vertices used to determine the shape of the outmost membrane #
  # of the SP systems located on each point of the lattice    #
  vertices
    (b1/2,b2/2)
    (-b1/2,b2/2)
    (-b1/2,-b2/2)
    (b1/2,-b2/2)
```

---

Chapter 2. Getting Started
endVertices

# Vectors pointing at the neighbours of each point of the lattice#
neighbours
  (1,0) (1,1) (0,1) (-1,1)
  (-1,0) (-1,-1) (0,-1) (1,-1)
endNeighbours

endLattice

A lattice is specified using the key words lattice ... endLattice and identified with a name, rectangularLattice in our example. The specification of a lattice consists of the following components:

1. Currently, our modelling language supports only one and two dimensional lattices. This must be specified after the key word dimension.

2. The lattice size is determined by the lower and upper bounds specified using the key words xmin, xmax, ymin and ymax followed by positive integer values.

3. A set of parameters can be used in the lattice specification. These are introduced using the key words parameters ... endParameters. Each parameter is identified with a name and is given a value.

4. The points of a lattice are determined by a set of basis vectors that are listed within the key words basis ... endBasis. The lattice points are then obtained as all the possible linear combinations of the basis vectors with integer coefficients within the given bounds.

5. A regular polygon, a rectangle in our example, is associated with each lattice point to specify the shape of the cell located in that position. The vertices of the polygon are computed using a list of vectors introduced using the key words vertices ... endVertices.

6. A neighbourhood is associated with each lattice point. This is specified as a list of vectors within the key words neighbours ... endNeighbours.

A model of a multi-cellular system in our modelling language consists of a Lattice Population P-system, LPP-system for short, a distribution of many clones of the different cell types represented as SP-systems over lattice points. These models must be specified in text files with the extension lpp. Open the model of our multi-cellular system consisting of three bacterial colonies by selecting File -> Open text file from the upper menu bar and choosing the file bacterialColonies.lpp:

```plaintext
# Author: Francisco J. Romero-Campero
# Date: July 2010
# Description: A multicellular system consisting of three bacterial colonies
# combining bacteria carrying the same gene under three different regulatory mechanisms. Namely, unregulated expression, positive autoregulation and negative autoregulation
LPPsystem threeColonies

# Cell types specified as individual SP systems
SPsystems
  SPsystem UnReg from UnReg.sps
  SPsystem PAR from PAR.sps
  SPsystem NAR from NAR.sps
endSPsystems

# The geometry of the system is represented using a regular finite point lattice
lattice rectangular from rectangular.lat

# Special distribution of the cells over the lattice points#
```

2.2. Tutorials
A multi-cellular model is specified using the key words \textit{LPPsystem} ... \textit{endLPPsystem} and is identified with a name, \textit{threeColonies}, in our example. The specification of a multi-cellular model consists of the following components:

1. A list of cell types defined as SP-systems introduced within the key words \textit{SPsystems} ... \textit{endSPsystems}. Each cell type is specified using the key word \textit{SPsystem} followed by the name given to the cell type, the key word \textit{from} and the file where the single cell model is specified. Recall that single-cell models can be specified in \textit{sps} format as described above or in \textit{SBML} format. Our autoregulation example uses three different cell types \textit{UnReg}, \textit{PAR} and \textit{NAR} introduced in the files \textit{UnReg.sps}, \textit{PAR.sps} and \textit{NAR.sps} respectively.

2. The declaration of the geometry of the system uses the key word \textit{lattice} followed by a name, the key word \textit{from} and the file where the corresponding \textbf{finite point lattice} is specified. Our example uses a rectangular lattice described in the \textit{rectangular.lat} file.

3. The \textbf{spatial distribution} of cellular clones over the lattice points. This is specified using the key words \textit{spatialDistribution} ... \textit{endSpatialDistribution}. A list of \textit{positions} associated to each cell type is introduced using the key words \textit{positions for} ... \textit{endPositions} specifying the name of the corresponding SP-system. These po-
positions can use some parameters with specific ranges described using the key word parameter followed by the parameter name, the equal symbol and the lower bound, step and upper bound separated by : symbols. Finally the coordinates, \( x \) and \( y \), are specified as mathematical expressions that may use the previously introduced parameters.

Below you have a figure representing the autoregulation model with its specific cell types and spatial distribution.

Multi-compartmental Stochastic Simulations

The Infobiotics Workbench allows you to perform stochastic simulations of your models. In this tutorial we will use our running example based on autoregulation to illustrate this feature. Alternatively, you can see our video tutorial.

Click on the Simulation tab located on the upper menu bar of the infobiotics dashboard to start up the dialog window below that will allow you to specify your simulation parameters.

In order to run your simulations you need to provide the following parameters:

1. First you need to specify your working directory. Click on the Browse button at the top right corner of the dialog window and navigate to the folder where the files comprising the autoregulation model are located.

2. Model file: You need to specify the file containing the multi-cellular model, bacterialColonies.lpp in our example. Click on the Browse button next to the Model file box and choose the corresponding lpp file.

3. Max time: The time you want your model to be simulated for must be specified in the corresponding box. For our example, please type 600.

4. Log interval: You need to set how often you want to save the state of your system in your simulation file. Please type 5 for our example.

5. Runs: The number of simulation runs to perform must be introduced in this box. In our example it is enough to run 100 simulations.

6. Data file: You must specify the name of the output file where your simulations will be saved. For instance, you can choose autoregulation_simulations.h5 for our case study.

The rest of the input parameters will be set to their default values in our example. Your simulation dialog window should look similar to the one below:

Other simulations parameters involving output and some spatial properties can be specified. Although they are not relevant in our running example they can be of interest for your models, see our documentation for more details.

You can save your simulation parameters by clicking on the Save button located on the upper menu bar of the simulation dialog window. These can be loaded in order to reproduce your simulation settings by clicking on the Load button and choosing the file containing your parameters.

In order to run your simulations click on the Perform button located at the bottom of the window. A progress bar will pop up to inform you that the simulations are running correctly. Once the simulations are over the following tab will appear on the main window to allow you to plot the results.

In our running example we can see the average evolution of protein1 molecules over time in three bacteria from the different colonies representing gene unregulated expression, positive autoregulation and negative autoregulation. In

---

2.2. Tutorials
Infobiotics Workbench, Release 0.0.1

Chapter 2. Getting Started

[Image of the Infobiotics Workbench GUI showing input and simulation settings.]
Directory: /home/clima/Desktop/autoregulation
Model file: bacterialColonies.lpp
Just initialise P system: 

**Simulation**

Max time: 600.0
Log interval: 5.0
Runs: 100
Data file: autoregulation_simulation.h5

Compress output: 
Compression level: 0
Simulation algorithm: Direct Method with queue
Random seed: 0

Loaded '/home/clima/Desktop/autoregulation/simulation_parameters_test.params'.
order to obtain a graph with this information choose **All** from the **Runs panel** on your left, **protien1** from the **Species panel** at the center and three bacteria (for example **NARbacterium (30,5)**, **PARbacterium (21,10)** and **UnRegbacterium (4,3)**) from the **Compartments panel** on your right. Finally, click on the first button located at the bottom right corner of the plotting window. A window with three different graphs will appear. You can combine these graphs to produce the figure below by selecting them (holding the key Ctrl) and clicking on the **Combine** button.

![Graph of protein1 molecules over time](image)

You can also see the spatio-temporal evolution of the number of **protein1** molecules over the entire multi-cellular system for a simulation by selecting a specific run from the **Runs panel** (for example, simulation 1), **protien1** from the **Species panel** and **All** from the **Compartments panel**. Finally, click on the second button located at the bottom right corner of the plotting window. A window will appear to show the spatio-temporal dynamics of **protein1** click on the **Play** button to see an image similar to the one below:

Check our [video](#) to see the spatio-temporal evolution of the number of **protein1** molecules.

### Model Formal Analysis using Model Checking

The Infobiotics Workbench allows you to analyse **probabilistic spatio-temporal properties** of your models using two model checkers PRISM and MC2. In this tutorial we will use our running example based on autoregulation to illustrate

2.2. Tutorials
Click on the **Model checking (PRISM)** tab located on the upper menu bar of the *infobiotics dashboard* to start up the dialog window below that will allow you to specify your model properties and the necessary parameters.

First of all, you need to specify your **working directory**. Click on the first **Browse** button at the top right corner of the dialog window and navigate to the folder where the files comprising the *autoregulation model* are located. Next you need to introduce the name of the file containing your P system model (lpp file). In our running example we will analyse the behaviour of single cells carrying three different gene regulatory mechanisms, gene unregulated expression, positive autoregulation and negative autoregulation. This model is specified in the file *individualCells.lpp*, please click on the corresponding **Browse** button and select this file.

Infobiotics workbench allows you to perform different **Tasks** in order to analyse properties of your model. The first task you must perform is to **Translate** your model into the specific language used by the model checker PRISM. In order to do this you need to provide a name for the PRISM model. Please type *autoregulation.sm* in the corresponding box. This model will be created after you click on the **Perform** button located at the bottom of the model checking dialog window. The PRISM model can be inspected by clicking on the **View** button next to the PRISM model box. You should see the following windows:

Typically, a PRISM model has a set of parameters representing the upper and lower bounds for the number of the different molecular species in the system. The stochastic constants associated with rules can also be model parameters. The values for these parameters must be specified using the **Model parameters** tab in the model checking dialog window and the box enumerating the **Model constants**. Notice that a brief description of each parameter is provided to assist you in choosing appropriate values for them. For our case study, please click on the **Value** box to set all the lower bounds for the different number of molecules to 0, the upper bounds for gene1 and protein1_gene1 to 1 and the rest of upper bounds to 1000.

The properties to be analysed can be specified using the **Temporal Formulas** tab. Click on it and give a name to the file where the properties will be saved, for example type *properties.csl*. New formulas can be added by clicking on the **Add temporal formula** button. Existing formulas can be modified by clicking on the **Edit temporal formula** or deleted by clicking of the **Remove temporal formula** button.

When specifying or editing the formulas that represent your model properties the following window will pop up. This
window allows you to specify/edit your formula, add/edit/remove parameters used in your properties and insert model parameters in your formula using the corresponding drop down list and Insert button. In our example, we analyse the response time (time needed to reach half the maximal expression of gene) of the different regulatory mechanisms represented in the autoregulation case study. The temporal formulas associated with the response time computes the probability of the number of protein1 molecules at positions (0,0), (1,0) and (2,0) in the compartments NARbacterium, PARbacterium and UnRegbacterium to exceed 50 molecules at time T where this parameter varies from 0 to 400 with a step of 1 as the following figure shows:

For more details on how to specify your model properties using temporal formulas you can visit the PRISM web site.

In our case study we will approximate the probabilities associated with the above temporal formulas. For this, please choose Approximate from the drop down list specifying the different Tasks that can be performed. Note that Verify is also available to analyse properties, nevertheless this task is computationally very expensive and is only feasible for very small systems. You also need to specify the Number of samples or simulation runs of your model that will be used in the approximation of the probabilities associated with your model properties. Please type 1000 runs for our example.

Finally, the Results file containing the output of the model checking analysis must be provided, autoregulation_results.sm for our example. Your model checking dialog window should look similar to the figure below:

Other parameters such as the states/transitions files and precision/confidence of the approximation can be specified. Although they are not relevant in our running example they can be of interest for your analysis, see our documentation for more details.

You can save your model checking parameters by clicking on the Save button located on the upper menu bar of the model checking dialog window. These can be loaded in order to reproduce your model checking settings by clicking on the Load button and choosing the file containing your parameters.

In order to run the analysis of your model properties according to the parameters you have introduced click on the
Perform button located at the bottom right corner. Once the model checking finishes the following tab appears on the main window to allow you to see the results for your different model properties.

A graph is generated for each temporal formula representing your model properties. These graphs can be detached so you can see the results from the different properties one next to the other by clicking on the Detach button at the bottom right corner.

The Infobiotics Workbench also allows you to analyse properties of your models using MC2. In this case you are able to reuse previous simulations performed for your model. Please click in the link below for a short tutorial on how to use the model checker MC2 integrated in Infobiotics Workbench.

**Model Checking with MC2**

Click on the Model checking (MC2) tab located on the upper menu bar of the infobiotics dashboard to start up the dialog window below that will allow you to specify your model properties and the necessary parameters to analyse them using MC2.

Similar to PRISM, you need to specify your working directory by clicking on the first Browse button of the dialog window and navigating to locatin of the autoregulation model. You also need to introduce the name of the file containing your P system model (lpp file), individualCells.lpp in our example. Please click on the corresponding Browse button and select this file.

MC2 allows you to reuse previously performed simulations of your models. In order to illustrate this, we have included in the folder containing the autoregulation model 5000 simulations in mc2 format in the file simulations_for_mc2.mc2. Please tick the Generated? box next to the MC2 input file box and choose the file simulations_for_mc2.mc2 by clicking on the Browse button in order to use the simulations contained in the file. These simulations can also be
2.2. Tutorials
generated using mcss, the simulator integrated in Infobiotics workbench, by providing in the corresponding box the
name for the simulation output file from mcss and the corresponding parameters by clicking on the Edit button.

You also need to provide in the box Number of samples the number of simulation runs to be used by MC2. Similar to
model checking with PRISM your model properties must be specified as temporal formulas. You can introduce your
model properties by clicking on the Add temporal formula button or by providing the name of the file containing
them using the corresponding Browse button. For our example, please choose the file formulas.pctl. Finally, you have
to enter the name of the Results file where the computed probabilities will be stored.

Your model checking dialog window should look similar to the one below:

You can save your parameters by clicking on the Save button located on the upper menu bar of the dialog window.
These can be loaded in order to reproduce your analysis settings by clicking on the Load button and choosing the file
containing your parameters.

In order to analyse your model properties click on the Perform button located at the bottom right corner. A progress
bar will pop up to inform you that the process is running correctly. Once the model checking is over the following tab
will appear on the main window to show you the results.

Model Structure and Parameter Optimisation

The Infobiotics Workbench allows you to optimise the structure and parameters of your models using evolutionary
algorithms. In this tutorial we will the example based on negative autoregulation to illustrate this feature. Alternatively,
you can see our video tutorial.

Click on the Optimisation tab located on the upper menu bar of the Infobiotics dashboard to start up the dialog window
below that will allow you to specify the optimisation parameters.

In order to optimise your models you need to provide the following parameters:

1. First you need to specify your working directory. Click on the Browse button at the top right corner of the
dialog window and navigate to the folder where the files comprising the negative autoregulation model are
located.

2. Number of different initial conditions: Specifies the number of different initial conditions. This number
should match the number of different initial and target files. In our example you can leave the default value 1.

3. Prefix for initial conditions filenames: You also need to specify the filename prefix that contains the initial
conditions for the objects. For example, setting the prefix initial, the files should be named as initial1.txt,
initial2.txt, and so on (as many different initial conditions you want to consider). Please type initial_values_NAR
for our example.

4. Number of target timeseries: Specifies the number of different target timeseries. This number should match
the number of columns (not counting with the time column) in the target files. For the example, type 2 (which
correspond to ‘protein’ and ‘rna’).

5. Prefix for target timeseries filenames: You need to specify the prefix for the filenames containing the target
times series. For example, setting the prefix target, the files should be named as target1.txt, target2.txt, and so
on (as many different initial conditions you want to consider). For our example, please type NAR_target.

6. Non-fixed module library: Here you need to insert the filename that contains the library of modules that are
variable i.e. that can be instantiated with different objects, kinetic constants, and compartments. For the example
use basicLibrary.plb.

7. Fixed model library: Specifies the filename that contains a library of fixed modules that should be in every
candidate model. For our example there is no need to specify such library, therefore the default value Null is
adequate.

8. Molecule names (colon-separated): Specifies all the molecules that can instantiate the modules. You can
simply name / for the single molecule in this example.
Infobiotics Workbench, Release 0.0.1

34 Chapter 2. Getting Started
9. **Random seed**: Introduces the seed for the random generator in optimisation. You can leave it to the default value 0 to set the seed based on the current time of your machine, or arbitrarily set it to a number.

The **Input parameters** tab in the optimisation window should look like this:

You can now select the **Evaluation** tab to specify the parameters related to the evaluation and optimisation of candidate models:

1. **Max time**: Indicates the time you want to simulate your model. Here, set this value to 360.
2. **Log interval**: Sets how often you want to save the state of your system in your simulation file. For the example, introduce the value 10. **NOTE**: The max time and log interval should be consistent with the target timeseries.
3. **Ensemble size (simulation runs)**: Number of stochastic simulations performed to evaluate each candidate model. This parameter can set to the default value 20 when running the example.
4. **Fitness function**: The fitness function determines how to measure the quality of the candidate models. You can simply use the default **Random-weight sum** method.
5. In the **Structure optimisation with Genetic Algorithm box**, you can specify some of the algorithm parameters for structure optimisation, such as:
   - **Maximum number of modules in a model**: Indicates the maximum number of modules a candidate model can contain. For the example, please type 3.
   - **Population size**: Number of candidate models in the GA population. For the example, choosing 5 should be enough.
   - **Number of generations**: Number of GA iterations. Please type 10, which should give enough time to the GA to find the target model.
6. In the **Parameter optimisation box**, you can specify some of the parameters for parameter optimisation, such as:
   - **Optimisation algorithm**: Determines which method should be used for parameter optimisation. You can choose between different evolutionary algorithms or simply go with the default method **Genetic Algorithm**.
Infobiotics Workbench, Release 0.0.1

Chapter 2. Getting Started

Please ensure the current working directory is correct.

Directory: /home/clima

Input parameters

- Number of initial model files: 1
- Prefix for initial model file names:
- Number of target timeseries: 1
- Prefix for target timeseries file names:
- Non-fixed module library: Browse...
- Fixed module library: Null
- Molecule names (colon-separated):
- Random seed: 0
• **Proportion of population to optimise**: Defines the proportion of candidate models that go under parameter optimisation. Ideally, all candidate models should have optimised parameters, but for computational reasons sometimes the user may choose to only optimise a certain best proportion of the population. For the example, you can set this value to 0.4.

• **Population size**: Number of candidate parameter sets in the population. For the example, choosing 5 should be enough.

• **Number of generations**: Number of iterations taken by the parameter optimisation method. For the example, choosing 10 should be sufficient to find the appropriate parameters.

Your simulation dialog window now should look similar to the one below:

You can save your optimisation parameters by clicking on the Save button located on the upper menu bar of the optimisation dialog window. These can be loaded in order to reproduce your optimisation settings by clicking on the
Load button and choosing the file containing your parameters.

In order to run the optimisation click on the Perform button located at the bottom right corner. A progress bar will pop up to inform you that the simulations are running correctly. Once the optimisation has finished (it should take around two minutes) the following tab will appear on the main window to allow you to visualise the behaviour of the best model found against the target model. You can also inspect the model structure and parameters.

For a complete description of the different components of Infobiotics Workbench please read our documentation.

2.2.2 Tutorial Using SBML Model Specification

Genes are a good example of biological switches. A cell can contain hundreds of thousands of genes, each of which can be switched on or off in response to internal or external signals. One important aspect of genetic switches (and switches in general) is how quickly they can be switched on. This is known as the response time of the gene. This tutorial examines the response times of three models of gene regulation, all representing mechanisms found in biological cells. You will learn how to implement these three models and perform stochastic simulations in order to quantify their response times.

A simple model of gene regulation

We first look at the simple model of gene regulation pictured below.
In this model a gene $G$ (yellow box) produces proteins $P$ (green box), which are also degraded. The expression of gene $G$ can be repressed by the negative transcription factor $R$ (green box), which binds with $G$ to form a complex $R.G$ (grey box containing $R$ and $G$ boxes). This prevents the gene $G$ from producing proteins. The action of the negative transcription factor $R$ can itself be inhibited by a protein $I$ (green box), which binds to $R$, forming a complex $R.I$ (grey box containing $R$ and $I$ boxes) and so prevents $R$ binding to the gene $G$.

This model can be summarised by the P system reaction scheme:

\[
\begin{align*}
G &\rightarrow G + P \\
G &\rightarrow P \\
R + G &\rightarrow R.G \\
R.G &\rightarrow R + G \\
R + I &\rightarrow R.I \\
R.I &\rightarrow R + I
\end{align*}
\]

For each reaction, a stochastic reaction constant $k$ is given. This reaction constant represents the average probability that a reaction will occur in an infinitesimal time interval. Since we are dealing with only one compartment, we don’t explicitly define the compartment.

We will perform a stochastic simulation of this model of genetic decision making and analyse its behaviour. In order to do this we need to parameterise the model i.e. assign values to the reaction constants and initial numbers of molecules for each protein and gene. Initially, we use the biologically naive set of parameter values given in the following table.

<table>
<thead>
<tr>
<th>reaction constants</th>
<th>initial molecules</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1 = 0.1$</td>
<td>$G_0 = 1$</td>
</tr>
<tr>
<td>$k_2 = 0.01$</td>
<td>$P_0 = 0$</td>
</tr>
<tr>
<td>$k_3 = 1.0$</td>
<td>$R_0 = 1$</td>
</tr>
<tr>
<td>$k_4 = 1.0$</td>
<td>$R.G_0 = 0$</td>
</tr>
<tr>
<td>$k_5 = 1.0$</td>
<td>$I_0 = 1$</td>
</tr>
<tr>
<td>$k_6 = 1.0$</td>
<td>$R.I_0 = 0$</td>
</tr>
</tbody>
</table>

Now we perform stochastic simulation of the parameterised model. In order to assess the average behaviour of the system we perform 1,000 runs of the stochastic simulation and average the number of molecules over these 1,000 runs. The following figure shows the average number of protein molecules $P$.

As mentioned before, a common measure used to quantify the behaviour of a gene regulation network is response time. The response time is defined as the time taken to reach half the steady state concentration. From the above figure
we can see that the steady state level of protein $P$ is on average around 6.6 molecules. Therefore, the response time is the time taken to reach half this level i.e. 3.3 molecules, which is around 65 seconds.

**Network motifs**

A *network motif* is a recurring pattern in a network that occurs far more often than at random. The simple regulation network above in one such motif. Two other common motifs found in gene regulation networks are negative autoregulation and positive autoregulation. Schematic representations of these three motifs are shown below.

In negative autoregulation, the expressed protein $P$ represses its own expression i.e. the $R$ proteins in the simple regulation model are changed to $P$.

$$
G \xrightarrow{k_1} G + P \quad P + G \xrightarrow{k_2} P.G \quad P + I \xrightarrow{k_5} P.I \\
G \xrightarrow{k_1} \emptyset \quad P.G \xrightarrow{k_4} P + G \quad P.I \xrightarrow{k_6} P + I
$$

In positive autoregulation, the expressed protein $P$ enhances its own expression i.e. the $I$ proteins in the same model are changed to $P$.

The model parameters are the same as those for the simple regulation model given above.

**Hands on**

You now proceed, step-by-step, to build such models in CellDesigner and simulating them with Infobiotics Workbench. Alternatively, you can download all the necessary files here and simply perform the simulation to observe the results.
Creating Models of Gene Regulation in CellDesigner

In this section we will show, step-by-step, how to create models using the CellDesigner package. Once we have these models, we will be able perform stochastic simulations on it and analyse the results.

Implementing a Simple Gene Regulation Model  
The model we will be creating first is the model of simple gene regulation described previously. The following tables summarize the reaction scheme and parameters for this model.

\[
\begin{array}{ccc}
G \xrightarrow{k_1} G + P & R + G \xrightarrow{k_2} R.G & R + P \xrightarrow{k_3} R.P \\
P \xrightarrow{k_4} \phi & R.G \xrightarrow{k_5} R + G & R.P \xrightarrow{k_6} R + P \\
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{reaction constants} & \text{initial molecules} \\
\hline
k_1 = 0.1 & G_0 = 1 & P_0 = 0 \\
k_2 = 0.01 & k_3 = 1.0 & R_0 = 1 \\
k_4 = 1.0 & k_5 = 1.0 & R.G_0 = 0 \\
k_6 = 1.0 & k_6 = 1.0 & I_0 = 1 \\
\hline
\end{array}
\]

The following figure shows an annotated screenshot of CellDesigner with the complete model.

Follow the steps below and you should end up with something very similar. As you work through the steps, don’t forget to save your model on a regular basis. Also, undo (Edit->Undo on the menubar or Ctrl-Z) can be very useful sometimes!

Step-by-step instructions:

1. Open up CellDesigner (we’re using version 4 here). You should find an icon on your desktop, or look in the Programs menu.

2. Create a new model by selecting File->New from the menubar. In the dialogue box that appears, enter a name for the model. The name should only contain alphanumeric characters and underscores for spaces. Since we’re creating a model of simple gene regulation, enter simple_regulation as the name, then click OK. If you want to change the model name or dimensions of the model canvas later, you can reopen this dialogue by selecting Component->Model Information from the menubar.

3. Save the model by selecting File->Save from the menubar (or pressing Ctrl-S) then clicking Save. As you work through the rest of the steps below, don’t forget to periodically save the model. Note: you might get a ‘libSBML Consistency Check’ warning when trying to save or load the model. Just ignore it and click Save.

4. Create the gene G by left clicking once on the Gene icon in the species toolbar then moving the pointer to the model canvas and left clicking on an empty space on the canvas. A dialogue box will appear asking for the name of the species. Enter G and click OK. A yellow rectangle containing the letter G should appear on the canvas. Species can be moved around the canvas by pointing to them, holding down the left mouse button and dragging them to a new location.

5. Set the initial amount of molecules to 1 for gene G by right clicking once on the newly created gene then choosing Edit Species from the menu that appears. In the dialogue box that pops up change 0.0 in the fourth text box (below the Amount radio button) to 1.0. Click Update then Close.
6. Create the protein $P$ by left clicking once on the **Generic Protein** icon in the species toolbar then moving the pointer to the model canvas and left clicking on an empty space on the canvas. A dialogue box will appear asking for the name of the species. Enter $P$ and click **OK**. A green rectangle containing the letter $P$ should appear on the canvas. Since, by default, all initial amounts of species are set to 0 we don’t need to set the initial amount of this species.

7. Create the reaction between the gene $G$ and protein $P$ by left clicking once on the **State Transition** icon in the reaction toolbar. Now move the pointer to gene $G$ on the canvas and left click once on one of the square anchor points which appears around the edge of gene $G$. Now move the pointer to protein $P$ on the canvas and left click once on one of the anchor points which appears around protein $P$. An arrow should appear between gene $G$ and protein $P$ representing the reaction between these two species. The point where each end of the arrow is anchored can be changed by left clicking once on the reaction arrow, moving the pointer to one of the anchor boxes which appears at each end of the arrow, holding down the left mouse button and dragging the anchor box to a new position.

8. Add gene $G$ as a product of the reaction you’ve just created between gene $G$ and protein $P$ by left clicking once on the **Add Product** icon in the reaction toolbar, then moving the pointer over the reaction arrow connecting gene $G$ and protein $P$. Left click the arrow once, then move the pointer to gene $G$ and left click one of the anchor boxes which appears around the edge of gene $G$. A second reaction arrow should appear from the arrow between gene $G$ and protein $P$ back to gene $G$.

9. Set the stochastic reaction constant for the reaction between gene $G$ and protein $P$ by pointing to the reaction arrow then right clicking it and selecting **Edit KineticLaw** from the menu that appears. Select the **Parameters** button from the dialogue box, then select **New**. In the dialogue box that pops up, enter $k1$ for the id and $0.1$ for the value. Then close all the dialogue boxes by clicking **Add** then **Close**, **Update**, **Close** and **Close**.

10. Add the degradation reaction for protein $P$ by left clicking once on the **Degradation** icon in the toolbar. Now move the pointer to protein $P$ on the canvas and left click the protein once. A reaction arrow and degradation symbol should appear attached to protein $P$. Before doing anything else, move back up to the toolbar and left click the **Select Move** icon to switch back to selection mode. As with any other species, you can move the pointer over the degradation symbol on the canvas, hold down the left mouse button, and drag it to a new position on the canvas.

11. Set the stochastic reaction constant for the degradation reaction for protein $P$ by pointing to the reaction arrow then right clicking it and selecting **Edit KineticLaw** from the menu that appears. Select the **Parameters** button from the dialogue box, then select **New**. In the dialogue box that pops up, enter $k2$ for the id and $0.01$ for the value. Then close all the dialogue boxes by clicking **Add** then **Close**, **Update**, **Close** and **Close**.

12. Create the repressor protein $R$ by left clicking once on the **Generic Protein** icon in the species toolbar then moving the pointer to the model canvas and left clicking on an empty space on the canvas. A dialogue box will appear asking for the name of the species. Enter $R$ and click **OK**. A green rectangle containing the letter $R$ should appear on the canvas.
13. Set the initial amount of molecules to 1 for repressor \( R \) by right clicking once on the newly created protein and choosing \textit{Edit Species} from the menu that appears. In the dialogue box that pops up change 0.0 in the fourth text box (below the \textit{Amount} radio button) to 1.0. Click \textit{Update} then \textit{Close}.

14. Create the repressor-gene complex by left clicking once on the \textbf{Complex} icon in the species toolbar then moving the pointer to the model canvas and left clicking on an empty space on the canvas. A dialogue box will appear asking for the name of the species. Enter \( R.G \) and click \textbf{OK}. A green rectangle containing the letters \( R.G \) should appear on the canvas. Now move the pointer back to the species toolbar and create a new repressor protein \( R \) by left clicking once on the \textbf{Generic Protein} icon in the species toolbar then moving the pointer to the model canvas and left clicking on an empty space on the canvas. A dialogue box will appear asking for the name of the species. Enter \( R \) and click \textbf{OK}. A green rectangle containing the letter \( R \) should appear on the canvas. Move the pointer over the new \( R \) protein, hold down the left mouse button, and drag the protein into the grey \( R.G \) complex box. Now create a new gene \( G \) by left clicking once on the \textbf{Gene} icon in the species toolbar then move the pointer to the model canvas and left click on an empty space on the canvas. A dialogue box will appear asking for the name of the species. Enter \( G \) and click \textbf{OK}. A yellow rectangle containing the letter \( G \) should appear on the canvas. Drag this gene into the grey \( R.G \) complex box. The complex box can be resized by left clicking on the black border of the grey complex box. A small white square should appear in each corner of the complex box. Move the pointer over one of these squares, hold down the left mouse button, and drag the mouse until the complex box is a reasonable size but still contains the \( R \) protein and \( G \) gene you dragged inside it.

15. Create the reaction which associates the repressor \( R \) and gene \( G \) into the \( R.G \) complex by left clicking once on the \textbf{Heterodimer Association} icon in the reaction toolbar. Now move the mouse first over the yellow gene \( G \) on the model canvas and left click once on one of the square anchor points which appears around the edge of the gene \( G \) box. Now move the pointer to the green repressor \( R \), and again left click once on one of the anchor points which appears. Finally, move the pointer to the edge of the grey \( R.G \) complex box and click on one of the anchor points that appears. A reaction arrow coming from the gene \( G \) and repressor \( R \) and leading to the \( R.G \) complex should appear.

16. Set the stochastic reaction constant for the association reaction between gene \( G \), repressor \( R \) and complex \( R.G \) by pointing to the reaction arrow then right clicking it and selecting \textit{Edit KineticLaw} from the menu that appears. Select the \textbf{Parameters} button from the dialogue box, then select \textbf{New}. In the dialogue box that pops up, enter \( k3 \) for the \textit{id} and 1.0 for the \textit{value}. Then close all the dialogue boxes by clicking \textit{Add} then \textit{Close}, \textit{Update}, \textit{Close} and \textit{Close}.

17. Create the reaction which dissociates the complex \( R.G \) into the repressor \( R \) and gene \( G \) by left clicking once on the \textbf{Dissociation} icon in the reaction toolbar. Now move the mouse first over the grey \( R.G \) complex box on the model canvas and left click once on one of the square anchor points which appears around the edge of the \( R.G \) complex box. Now move the pointer to the yellow gene \( G \) box, and again left click once on one of the anchor points which appears. Finally, move the pointer to the green repressor \( R \) box and click on one of the anchor points that appears. A reaction arrow coming from the \( R.G \) complex and leading to the gene \( G \) and repressor \( R \) boxes should appear.

18. Set the stochastic reaction constant for the dissociation reaction between complex \( R.G \) and gene \( G \) and repressor \( R \) by pointing to the reaction arrow then right clicking it and selecting \textit{Edit KineticLaw} from the menu that appears. Select the \textbf{Parameters} button from the dialogue box, then select \textbf{New}. In the dialogue box that pops up, enter \( k4 \) for the \textit{id} and 1.0 for the \textit{value}. Then close all the dialogue boxes by clicking \textit{Add} then \textit{Close}, \textit{Update}, \textit{Close} and \textit{Close}.

19. Create the inhibitor protein \( I \) by left clicking once on the \textbf{Generic Protein} icon in the species toolbar then moving the pointer to the model canvas and left clicking on an empty space on the canvas. A dialogue box will appear asking for the name of the species. Enter \( I \) and click \textbf{OK}. A green rectangle containing the letter \( I \) should appear on the canvas.

20. Set the initial amount of molecules to 1 for inhibitor \( I \) by right clicking once on the newly created protein and choosing \textit{Edit Species} from the menu that appears. In the dialogue box that pops up change 0.0 in the fourth text box (below the \textit{Amount} radio button) to 1.0. Click \textit{Update} then \textit{Close}.

21. Create the repressor-inhibitor complex by left clicking once on the \textbf{Complex} icon in the species toolbar then
moving the pointer to the model canvas and left clicking on an empty space on the canvas. A dialogue box will appear asking for the name of the species. Enter R.I and click OK. A grey rectangle containing the letters R.I should appear on the canvas. Now move the pointer back to the toolbar and create a new repressor protein R by left clicking once on the Generic Protein icon in the species toolbar then moving the pointer to the model canvas and left clicking on an empty space on the canvas. A dialogue box will appear asking for the name of the species. Enter R and click OK. A green rectangle containing the letter R should appear on the canvas. Move the pointer over the new R protein, hold down the left mouse button, and drag the protein into the grey R.I complex box. Now create a new inhibitor I by left clicking once on the Generic Protein icon in the species toolbar then moving the pointer to the model canvas and left clicking on an empty space on the canvas. A dialogue box will appear asking for the name of the species. Enter I and click OK. A green rectangle containing the letter I should appear on the canvas. Drag this inhibitor into the grey R.I complex box. The complex box can be resized as before.

22. Create the reaction which associates the repressor R and inhibitor I into the R.I complex by left clicking once on the Heterodimer Association icon in the reaction toolbar. Now move the mouse first over the green repressor R box on the model canvas and left click once on one of the square anchor points which appears around the edge of the repressor R box. Now move the pointer to the green inhibitor I box, and again left click once on one of the anchor points which appears. Finally, move the pointer to the edge of the grey R.I complex box and click on one of the anchor points that appears. A reaction arrow coming from the repressor R and inhibitor I and leading to the R.I complex should appear.

23. Set the stochastic reaction constant for the association reaction between repressor R, inhibitor I and complex R.I by pointing to the reaction arrow then right clicking it and selecting Edit KineticLaw from the menu that appears. Select the Parameters button from the dialogue box, then select New. In the dialogue box that pops up, enter k5 for the id and 1.0 for the value. Then close all the dialogue boxes by clicking Add then Close, Update, Close and Close.

24. Create the reaction which dissociates the complex R.I into the repressor R and inhibitor I by left clicking once on the Dissociation icon in the reaction toolbar. Now move the mouse first over the grey R.I complex box on the model canvas and left click once on one of the square anchor points which appears around the edge of the R.I complex box. Now move the pointer to the green repressor R box, and again left click once on one of the anchor points which appears. Finally, move the pointer to the green inhibitor I box and click on one of the anchor points that appears. A reaction arrow coming from the R.I complex and leading to the repressor R and inhibitor I boxes should appear.

25. Set the stochastic reaction constant for the dissociation reaction between complex R.I, and repressor R and inhibitor I by pointing to the reaction arrow then right clicking it and selecting Edit KineticLaw from the menu that appears. Select the Parameters button from the dialogue box, then select New. In the dialogue box that pops up, enter k6 for the id and 1.0 for the value. Then close all the dialogue boxes by clicking Add then Close, Update, Close and Close.

26. Export the model as SBML Level 2 by selecting File->Export Pure Level 2 Version 1 from the menubar and replacing untitled by simple_regulation.sbml in the Selection text box, then click Save.

Congratulations - you now have a model of simple gene regulation which is ready to simulate!

Implementing negative and positive gene regulation  We now implement the other two models (negative autoregulation and positive autoregulation). The reaction schemes are

\[
\begin{align*}
G & \xrightarrow{k_1} G + P \\
G & \xrightarrow{k_2} P.G \\
\phi & \xrightarrow{k_3} P + G \\
P.I & \xrightarrow{k_4} P + I
\end{align*}
\]

for the negative autoregulation and
for the positive autoregulation. You don’t need to redraw each model. Instead, save the simple regulation model you implemented before under a different name e.g. negative_autoregulation (File->Save As in the CellDesigner menubar), edit the model information (Component->Model Information in the menubar), and then rename the appropriate species (right click the species and select Change Identity).

Simulating Models with Infobiotics Workbench

In this section we will show, step-by-step, how to perform stochastic simulations of models and analyse the results of these simulations.

Simulating simple gene regulation We will start by using the model of simple gene regulation implemented before in CellDesigner. For this model, we will show how to calculate the response time of this gene regulatory network. To simulate the model, we will use the Infobiotics Workbench (IBW). We now take a look at some screenshots of IBW.

The simulation control window of IBW where the parameters for the simulation are entered:

The simulation results tab, which opens automatically once the simulation has finished:

The simulation results tab is used to select the runs, species and compartments which you want to plot graphs of. Once these have been selected, a graph overview window will open, which will show and allow further manipulation of these graphs:

The final graph you should obtain from which you will be able to calculate the response time is:

Follow the steps below and you should end up with something very similar:

1. Open the simulation window by clicking on Simulation.
2. Set the name of the model file in the Model file text box in the simulation control window. Since we are simulating the simple gene regulation model implemented before, you should enter simple_regulation.sbml for the name of model file.
3. Set the time in seconds you want to simulate the model for. We will be simulating each simulation run for 20 minutes i.e. 1200 seconds, so enter 1200 in the Max time text box. Set also the Log interval to 1.
4. Set the number of simulation runs of the model you want to perform. With stochastic simulation, due to the random fluctuations inherent in this modelling approach, it is common to perform a number of simulation runs and average the number of molecules of each species over these runs to gain an insight into the average behaviour of the model. We will do this for the simple regulation model and average the species levels over 1,000 runs, so enter 1000 in the Runs text box.
5. Set the name of the data file you want to save the simulation results in. Simulation results are stored in HDF5 format (a scientific data storage standard), whose files usually use the ‘.h5’ extension. Since we are simulating the simple gene regulation model, in the Data file text box enter simple_regulation.h5 for the data file filename.
6. Leave the rest of the parameters with their default values.
Infobiotics Workbench, Release 0.0.1

2.2. Tutorials

![Image of Infobiotics Workbench interface with parameters for model simulation and data loading.](image-url)
7. Simulate the model by clicking on the **Perform** button. A progress bar should appear showing the progress of the simulation. Once all the simulation runs have finished (this may take around a minute) the simulation results tab should appear.

8. Select the runs you want to use when plotting graphs. In the simulation results window there are three panes, the left hand one of which (titled **Runs**) lists the runs performed during the simulation. We want to use all these runs to calculate the average levels of species in the simulation. so check the **All** box at the top of this pane. All the runs in the list should now be highlighted.

9. Select the species you want to plot graphs for. In the middle pane (titled **Species**) of the simulation results window you’ll see a list of species in the simple gene regulation model. We are interested in the levels of the protein $P$ which is produced by the gene, so highlight $P$ in the list of species by left clicking it once.

10. Select the compartments you want to plot graphs for. In the right hand pane (titled **Compartments**) you’ll see a list of compartments. Since there is only one compartment in the simple gene regulation model there will only be one entry (called `compartment:1::0,0`) in this list. Highlight this entry in the list of compartments by left clicking it once.

11. Click on the first button located at the bottom right corner of the simulation tab. The graph overview window should now appear.

12. Open the plot of protein $P$ vs. time in a separate window by pointing to this graph in the graph overview window and left clicking it once to highlight it. Now click on the **Tile** button. A new window containing this graph should open. You can resize this window, zoom in and out, use the cursor to determine the location of points of the graph, and save the resulting graph.

13. Determine the response time of the simple gene regulation model. Remember, the response time is defined as the time taken to reach half the steady-state level. So, first determine the steady-state level by examining the graph of protein $P$ vs. time. Now examine the graph to find the time at which the level of protein $P$ reaches half
Select multiple plots (using Ctrl-click) to combine, stack or tile. Drag to rearrange.

Hide invariants (using Ctrl-A to select all always selects invariants)
this steady-state level.

**Congratulations** - you have now performed a stochastic simulation of a simple gene regulation network and calculated its response time!

**Simulating negative and positive autoregulation**  Simulate and determine the response times of the other two models of gene regulation (negative autoregulation and positive autoregulation).

You now should be able to answer the following questions:

1. Which of the three motifs (simple regulation, negative autoregulation, and positive autoregulation) gives the fastest response time and which one the slowest?
2. Can you explain why this is the case? (looking at the levels of the other species).
3. What other differences do you notice in the production of protein $P$ between the three models?

For a complete description of the different components of **Infobiotics Workbench** please read our documentation.

### 2.3 Documentation

**2.3.1 Model Specification and Building**

The specification and building of **multi-cellular system** models in **Infobiotics** is modular allowing parsimonious and incremental design. In this chapter, the **Infobiotics modelling language** is introduced using a running example consisting of the synthetic bacterial colony designed by Ron Weiss’ group in [Basu2005]. This model implements the propagation of a wave of gene expression in a bacterial colony.
The Infobiotics modelling language provides a multi-compartmental, stochastic and rule-based specification framework. A model of a multi-cellular system in Infobiotics is developed as a Lattice Population P-system (LPP-system) which consists of the specification of three main components that can be defined in a modular manner:

1. First, the different cell types in the multi-cellular system need to be specified including their molecular species, compartmentalised structure and molecular interactions.
2. Second, the geometric distribution of the cells in the multi-cellular system has to be captured using a finite point lattice, a regularly distributed collection of spatial points.
3. Finally, the specific localisation of the different cells over the lattice points must be described in order to obtain the final spatial distribution in the multi-cellular system.

![Figure 2.1: Example of an LPP-system consisting of the distribution of the model of a single cell containing a negative autoregulated gene (right) over a regular rectangular lattice (left).](image)

The specification of the different parts of a multicellular system using the Infobiotics modelling language is specified in the following sections:

### Specification of Cell Types

Individual cells are the elementary unit of our models. Nevertheless, they are not represented as homogeneous bags in our framework. Instead we use Stochastic P-systems (SP-systems) [Romero-Campero2009] as the computational abstraction for an individual cell. The specification of an individual cell consists of the following three main components:

1. The molecular species (genes, RNAs, proteins, signals etc.) present in the specific cell type are represented using string-objects like proteinGFP, signal3OC6, etc.
2. The compartments (cytoplasm, nucleus, mitochondria, etc) of individual cells as well as some relevant regions related to individual cells (cell surface, media surrounding a cell, etc.) are defined using membranes. The inclusion of compartments inside other compartments as in the case of the nucleus being inside the cytoplasm can also be specified.
3. The characteristic molecular interactions taking place inside or between specific compartments are described using rules where the reactant and product molecules are specified as well as the compartment involved in the interaction and a stochastic constant used to compute the probability of applying each rule and the time elapsed between rule applications.

The specification of the components of a model of an individual cell using the Infobiotics modelling language is described in the following sections:
**Single Cell Specification Format**

The specification of a single cell type can be provided to Infobiotics in SBML format. Specifically, Infobiotics is compatible with the SBML v2.1 generated by CellDesigner.

Alternatively, the specific modelling language based on SP-systems provided by Infobiotics can be used. The skeleton of the specification of an individual cell type, identified with the name `cellTypeName` must be done in a text file with extension `.sps` for *Stochastic P-system* according to the following format:

```
SPSystem cellTypeName
  alphabet
  ...  
  endAlphabet

  compartments
  ...  
  endCompartments

  initialMultisets
  ...  
  endInitialMultisets

  ruleSets
  ...  
  endRuleSets
```

Figure 2.2: Example of an *SP-system* model of a cell type with three compartments, cell surface, cytoplasm and nucleus, string-objects representing molecular species as receptors, signals and genes and molecular interactions specified as rules describing a signal transduction pathway.
In Ron Weiss’ synthetic pulse propagation, there exist two different bacterial strains or cell types. One of them acts a \textbf{signal sender} and the other one as a \textbf{pulse generator} in response to the signal produced by the first bacterial strain. In what follows we use the specification of the \textit{signal sender} to introduce the format for the specification of the different components of an SP-system model enumerated above.

**Molecular Species**  The molecular species described as string-objects are specified within the block \texttt{molecules ... endMolecules}. Each molecular species is identified with a name, \texttt{moleculeName}. For example, in our \textit{signal sender} the following molecular species are present:

```plaintext
alphabet
  Pconst_geneLuxI
  rnaLuxI_RNAP
  rnaLuxI
  proteinLuxI_Rib
  proteinLuxI
  signal3OC6
endAlphabet
```

These objects represent the constitutively expressed gene LuxI, \texttt{Pconst\_geneLuxI}, an RNA polymerase producing the RNA associated with LuxI, \texttt{rnaLuxI\_RNAP}, the LuxI RNA, \texttt{rnaLuxI}, a ribosome translating the corresponding RNA into the protein LuxI, \texttt{proteinLuxI\_Rib}, the LuxI protein, \texttt{proteinLuxI} and the molecule signal \texttt{3OC6}, \texttt{signal3OC6}.

**Compartments**  The different compartments and regions relevant in a model of an individual cell type are specified in the block \texttt{compartments ... endCompartments}. Each compartment is identified with a \texttt{compartmentName} and the keyword \texttt{inside} followed by another \texttt{compartmentName} is used to specify the inclusion of the first compartment in the second one. The declaration of a compartment identified with \texttt{compartmentName1} inside a compartment identified with \texttt{compartmentName2} is specified as follows:

```plaintext
compartmentName1 inside compartmentName2
```

If a compartment is not contained in any other compartment of the system the key word \texttt{inside} and the second compartment name are omitted.

Our example includes two relevant regions for a bacterium. Specifically, the \textit{media} surrounding an individual cell and the \textit{bacterium} itself. Note how the compartment \textit{bacterium} is declared as contained in the compartment \textit{media}:

```plaintext
compartments
  media
    bacterium inside media
endCompartments
```

**Initial Multiplicities**  The initial number of molecules present at the beginning of the simulation in the different compartments must be specified using the block \texttt{initialMultisets ... endInitialMultisets}. For each compartment identified with \texttt{compartmentName} a different block \texttt{initialMultiset compartmentName ... endInitialMultiset} must be declared inside the block \texttt{initialMultisets}. This inner block contains a list of species names follow by an integer representing the initial number of molecules:

```plaintext
initialMultisets
  initialMultiset compartmentName
    speciesName numberOfMolecules
    ...
endInitialMultiset
...
endInitialMultisets

If a compartment is initially empty the block declaring its initial number of molecules can be omitted. For example, our signal sender consists of two compartments, the media and the bacterium. The media is initially empty whereas the bacterium has a molecule $Pcons\_geneLuxI$ representing a specific gene:

```
initialMultisets
  initialMultiset bacterium
    $Pconst\_geneLuxI$ 1
  endInitialMultiset
endInitialMultisets
```

**Molecular Interactions**  The molecular interactions in a cell type are represented using rules specified within the block `ruleSets ... endRuleSets`. For each compartment identified with `compName` a different block `ruleSet compName ... endRuleSet` enumerating the molecular interactions associated with the compartment must be inserted in the previous block:

```
ruleSets
  ruleSet compName
    ...
  endRuleSet
...
endRuleSets
```

The sender cell in our example needs the specification of the molecular interactions involving two different compartments, the media and the bacterium:

```
ruleSets
  ruleSet media
    ...
  endRuleSet
  ruleSet bacterium
    ...
  endRuleSet
endRuleSets
```

The molecular interactions taking place inside a compartment (first rule type below) or moving molecules outside a compartment (second rule type below) and inside a compartment (third rule type below) are specified using one of the rule types below. These must be specified within the corresponding block `ruleSet compName ... endRuleSet`:

```
ruleName: [ reactants ]_compName -const-> [ products ]_compName const = value
ruleName: [ reactants ]_compName -const-> products [ ]_compName const = value
ruleName: reactants [ ]_compName -const-> [ products ]_compName const = value
```

In the above declaration `ruleName` is an identifier of the rule, `reactants` and `products` are either a single `moleculeName` or two separated by a plus symbol, `moleculeName + moleculeName`; the compartment involved in the molecular interaction is identified with `compName`; `const` is used to represent the stochastic constant specifically associated with the rule and its value is specified in `const = value`.

In the sender cell of our running example there are two different compartments, the media and the bacterium. In the media the only molecular interactions are signal diffusion out of the media, signal diffusion inside the bacterium and signal degradation. These are represented using the rules $r1$, $r2$ and $r3$ specified in the corresponding `ruleSet` block:
Infobiotics Workbench, Release 0.0.1

ruleSet media
    r1: [ signal3OC6 ]_media -c1-> signal3OC6 [ ]_media c1 = 1
    r2: signal3OC6 [ ]_bacterium -c2-> [ signal3OC6 ]_bacterium c2 = 2
    r3: [ signal3OC6 ]_bacterium -c3-> [ ]_bacterium c3 = 1
endRuleSet

The molecular interactions associated with a compartment can be specified in a modular way by reusing sets of rules, modules, containing variables that can be instantiated with specific molecular species names, stochastic constant values and compartment names. A module is specified by stating its identifier moduleName and the molecular names, stochastic constant values and compartment names for the corresponding instantiations (objectInstantiation, constantInstantiation, compInstantiation). Finally, the file containing the library of modules where the corresponding module is defined must be specified after the key word from:

moduleName( objectInstantiation, constantInstantiation, compInstantiation ) from moduleLibraryFile

For example, in our sender cell the molecular interactions associated with the bacterium compartment are specified using two modules defined in the module library library.plb. The first one called Pconst contains rules describing the constitutive expression of a gene X, instantiated in this case with LuxI, at a rate c, instantiated with 0.1, inside a compartment l, instantiated with bacterium here. The second one called PostTransc consists of rules representing the processes involved in the post-transcriptional regulation of a protein X, instantiated with LuxI here, taking place a the rates instantiated with the values 3.36,0.0667,0.004,3.78,0.0667 inside a compartment instantiated as bacterium in this case.

Three additional rules are added in order to represent signal synthesis by proteinLuxI, signal diffusion out of the compartment bacterium and signal degradation:

ruleSet bacterium
    Pconst({LuxI},{0.1},{bacterium}) from library.plb
    PostTransc({LuxI},{3.36,0.0667,0.004,3.78,0.0667},{bacterium}) from library.plb
    r1: [ proteinLuxI ]_bacterium -c1-> [ proteinLuxI + signal3OC6 ]_bacterium c1 = 1
    r2: [ signal3OC6 ]_bacterium -c2-> signal3OC6 [ ]_bacterium c2 = 2
    r3: [ signal3OC6 ]_bacterium -c3-> [ ]_bacterium c3 = 2
endRuleSet

The use of modules is extensively described in the next section.

Library of Molecular Interaction Modules

The Infobiotics Workbench supports the definition of libraries of molecular interaction modules which allows the user to specify the set of rules in a model of an individual cell in a modular manner. Our modules facilitate the hierarchical, incremental and parsimonious development of the specifications of the molecular interactions associated with SP-system models of individual cells as explained in the following sections:

Molecular Interaction Modules  A molecular interaction module consists of a set of rules representing a general schema that can be instantiated to produce specific molecular interactions. These rules may contain some variables for the name of the molecular species, stochastic constant values and compartment names so they can be reused intensively in our models by instantiating these variables with specific molecules, rates and compartments.

A module is identified with a name, moduleName and three lists of variables for molecular species, speciesVariables, for stochastic constants, constantVariables, and for compartment names, compartmentVariables. The specification of a module enumerates the rules, molecularInteractionRule, with variables used to represent the schema of molecular interactions. In this respect, if the stochastic constant associated with a rule is a variable of the module the last part of a rule specification (const = value) is omitted. The definition of a module is specified as follows:
moduleName(speciesVariables,constantVariables,compartentVariables) =
{
    molecularInteractionRule
    
    molecularInteractionRule
}

The set of rules associated with a module can be specified using other modules. In this case it is necessary to specify the file containing the library of modules where the module definition is declared:

moduleName(speciesVariables,constantVariables,compartentVariables) =
{
    moduleName_1(speciesVariables_1,constantVariables_1,compartentVariables_1) from moduleLibraryFile
    
    moduleName_n(speciesVariables_n,constantVariables_n,compartentVariables_n) from moduleLibraryFile
    
    molecularInteractionRule
    
    molecularInteractionRule
}

This facilitates the hierarchical and incremental design of complex modules and consequently, the parsimonious and modular specification of the molecular interactions associated with a SP-system model of a single cell. The modules used in the our sender cell are introduced below to illustrate the definition of modules:

Pconst({X},{c_1},{1}) =
{
    r1: [ Pconst_geneX ]_l -c1-> [ Pconst_geneX + rnaX_RNAP ]_l
}

PostTransc({X},{c_1,c_2,c_3,c_4,c_5},{1}) =
{
    r1: [ rnaX_RNAP ]_l -c_1-> [ rnaX ]_l
    
    r2: [ rnaX ]_l -c_2-> [ rnaX + proteinX_Rib ]_l
    
    r3: [ rnaX ]_l -c_3-> [ ]_l
    
    r4: [ proteinX_Rib ]_l -c_4-> [ proteinX ]_l
    
    r5: [ proteinX ]_l -c_5-> [ ]_l
}

**Libraries of Modules** Molecular interaction modules must be defined in libraries, files with the extension .plb, in order to facilitate their reusability in different SP-system models of individual cells. A library of modules is defined as a block libraryOfModules libraryName ... endLibraryOfModules where libraryName is an identifier. A library consists of a list of module definitions as introduced in the previous section. The general structure of a module library is given below:

libraryOfModules libraryName
    moduleDefinition
    
    moduleDefinition
endLibraryOfModules

Modules can reuse other modules define in the same library if these have been defined previously, above in the file, or modules from other libraries. When the modules being reused are from the same library the two keywords from this must be used, if they are defined in other libraries the corresponding files must be declared following the module instantiation using from libraryFileName.
The following next two sections illustrate how modules and libraries of modules are used in the Infobiotics Workbench in two different scenarios: systems biology and synthetic biology.

The library of modules used in our model of Ron Weiss’ pulse propagation is presented below in order to illustrate the above definition. Observe how the last module in the library pulseGenerator is defined using previously introduced modules in the library with specific instantiations. As illustrated in the first module, Pconst, a variable representing a stochastic constant can be given a default value that can be overwritten.

```
libraryOfModules promoterLibrary

Pconst((X),(c_1),(1)) =
{
    r1: [ Pconst_geneX ]_l -c_1-> [ Pconst_geneX + rnaX_RNAP ]_l  
    c1 = 2
}

Plux((X),(c_1, c_2, c_3),(1)) =
{
    r1: [ LuxR2 + Plux_geneX ]_l -c_1-> [ Plux_LuxR2_geneX ]_l
    r2: [ Plux_LuxR2_geneX ]_l -c_2-> [ LuxR2 + Plux_geneX ]_l
    r3: [ Plux_LuxR2_geneX ]_l -c_3-> [ Plux_LuxR2_geneX + rnaX_RNAP ]_l
}

Pluxleaky((X),(c_1, c_2, c_3, c_4),(1)) =
{
    r1: [ LuxR2 + Plux_geneX ]_l -c_1-> [ Plux_LuxR2_geneX ]_l
    r2: [ Plux_LuxR2_geneX ]_l -c_2-> [ LuxR2 + Plux_genex ]_l
    r3: [ Plux_LuxR2_geneX ]_l -c_3-> [ Plux_LuxR2_geneX + rnaX_RNAP ]_l
    r4: [ Plux_geneX ]_l -c_4-> [ Plux_geneX + rnaX_RNAP ]_l
}

PluxPR((X),(c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9),(1)) =
{
    r1: [ LuxR2 + PluxPR_geneX ]_l -c_1-> [ PluxPR_LuxR2_geneX ]_l
    r2: [ PluxPR_LuxR2_geneX ]_l -c_2-> [ LuxR2 + PluxPR_geneX ]_l
    r3: [ LuxR2 + PluxPR_CI2_geneX ]_l -c_3-> [ PluxPR_LuxR2_CI2_geneX ]_l
    r4: [ PluxPR_LuxR2_CI2_geneX ]_l -c_4-> [ LuxR2 + PluxPR_CI2_geneX ]_l
    r5: [ CI2 + PluxPR_geneX ]_l -c_5-> [ PluxPR_CI2_geneX ]_l
    r6: [ PluxPR_CI2_geneX ]_l -c_6-> [ CI2 + PluxPR_geneX ]_l
    r7: [ CI2 + PluxPR_LuxR2_geneX ]_l -c_7-> [ PluxPR_LuxR2_CI2_geneX ]_l
    r8: [ PluxPR_LuxR2_CI2_geneX ]_l -c_8-> [ CI2 + PluxPR_LuxR2_geneX ]_l
    r9: [ PluxPR_LuxR2_geneX ]_l -c_9-> [ PluxPR_LuxR2_geneX + rnaX_RNAP ]_l
}

Plac((X),(c_1, c_2, c_3),(1)) =
{
    r1: [ Plac_geneX ]_l -c_1-> [ Plac_geneX + rnaX_RNAP ]_l
    r2: [ proteinLacI + Plac_geneX ]_l -c_2-> [ Plac_LacI_geneX ]_l
    r3: [ Plac_LacI_geneX ]_l -c_3-> [ proteinLacI + Plac_geneX ]_l
    r4: [ proteinUnLacI + Plac_geneX ]_l -c_2-> [ Plac_UnLacI_geneX ]_l
    r5: [ Plac_UnLacI_geneX ]_l -c_3-> [ proteinUnLacI + Plac_geneX ]_l
}

PluxPtetR((X),(c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_10, c_11, c_12, c_13, c_14, c_15),(1)) =
{
    r1: [ LuxR2 + PluxPtetR_geneX ]_l -c_1-> [ PluxPtetR_LuxR2_geneX ]_l
    r2: [ PluxPtetR_LuxR2_geneX ]_l -c_2-> [ LuxR2 + PluxPtetR_geneX ]_l
    r3: [ LuxR2 + PluxPtetR_TetR_geneX ]_l -c_3-> [ PluxPtetR_LuxR2_TetR_geneX ]_l
    r4: [ PluxPtetR_LuxR2_TetR_geneX ]_l -c_4-> [ LuxR2 + PluxPtetR_TetR_geneX ]_l
}
```

2.3. Documentation 57
Infobiotics Workbench, Release 0.0.1

58 Chapter 2. Getting Started

r5: [ LuxR2 + PluxPtetR_TetR2_geneX ]_1 -c_5-> [ PluxPtetR_LuxR2_TetR2_geneX ]_1
r6: [ PluxPtetR_LuxR2_TetR2_geneX ]_1 -c_6-> [ LuxR2 + PluxPtetR_TetR2_geneX ]_1
r7: [ TetR + PluxPtetR_geneX ]_1 -c_7-> [ PluxPtetR_TetR_geneX ]_1
r8: [ PluxPtetR_TetR_geneX ]_1 -c_8-> [ TetR + PluxPtetR_TetR2_geneX ]_1
r9: [ TetR + PluxPtetR_LuxR2_TetR2_geneX ]_1 -c_9-> [ PluxPtetR_LuxR2_TetR2_geneX ]_1
r10: [ PluxPtetR_LuxR2_TetR2_geneX ]_1 -c_10-> [ TetR + PluxPtetR_LuxR2_TetR_geneX ]_1
r11: [ TetR + PluxPtetR_TetR2_geneX ]_1 -c_11-> [ PluxPtetR_TetR2_geneX ]_1
r12: [ PluxPtetR_TetR2_geneX ]_1 -c_12-> [ TetR + PluxPtetR_TetR_geneX ]_1
r13: [ TetR + PluxPtetR_LuxR2_TetR_geneX ]_1 -c_13-> [ PluxPtetR_LuxR2_TetR2_geneX ]_1
r14: [ PluxPtetR_LuxR2_TetR2_geneX ]_1 -c_14-> [ TetR + PluxPtetR_LuxR2_TetR_geneX ]_1
r15: [ PluxPtetR_LuxR2_geneX ]_1 -c_15-> [ PluxPtetR_LuxR2_geneX + rnaX_RNAP ]_1

PR({X},{c_1,c_2,c_3},{l}) =
{ r1: [ PR_geneX ]_1 -c_1-> [ PR_geneX + rnaX_RNAP ]_1
r2: [ CI2 + PR_geneX ]_1 -c_2-> [ PR_CI2_geneX ]_1
r3: [ PR_CI2_geneX ]_1 -c_3-> [ CI2 + PR_geneX ]_1
}

PostTransc({X},{c_1,c_2,c_3,c_4,c_5},{l}) =
{ r1: [ rnaX_RNAP ]_1 -c_1-> [ rnaX ]_1
r2: [ rnaX ]_1 -c_2-> [ rnaX + proteinX_Rib ]_1
r3: [ rnaX ]_1 -c_3-> [ ]_1
r4: [ proteinX_Rib ]_1 -c_4-> [ proteinX ]_1
r5: [ proteinX ]_1 -c_5-> [ ]_1
}

Dim({X,Y},{c_1,c_2},{l}) =
{ r1: [ proteinX + proteinX ]_1 -c_1-> [ Y ]_1
r2: [ Y ]_1 -c_2-> [ ]_1
}

DimSig({X,S,Y},{c_1,c_2,c_3,c_4,c_5},{l}) =
{ r1: [ proteinX + signalS ]_1 -c_1-> [ proteinX_S ]_1
r2: [ proteinX_S ]_1 -c_2-> [ ]_1
r3: [ proteinX_S + proteinX_S ]_1 -c_3-> [ Y ]_1
r4: [ Y ]_1 -c_4-> [ ]_1
}

Diffusion({X},{c_1},{l}) =
{ r1: [ signalX ]_1 = (1,0) = [ ] -c_1-> [ ]_1 = (1,0) = [ signalX ]
}

Deg({X},{c_1},{l}) =
{ r1: [ X ]_1 -c_1-> [ ]_1
}

pulseGenerator({X},{c_1,c_2,c_3,c_4,c_5},{l}) =
{ }
Libraries of Modules in Systems Biology  In a Systems biology scenario modules can represent basic regulatory mechanisms or regulatory motifs in cellular systems that appear recurrently involving different molecular species interacting according to characteristic rates in specific locations of the cells. For example, the following basic library defines modules representing positive, negative and constitutive gene expression as well as some basic transcriptional regulatory motifs in bacterial systems as negative autoregulation (NAR) and incoherent feedforward loop (IFFL):

libraryOfModules transcriptionalMotifs

Const((X),(c_1),(1)) =
{
  r1: [ geneX ]_l -c_1-> [ geneX + rnaX ]_l
}

PosReg((X,Y),(c_1,c_2,c_3),(1)) =
{
  r1: [ proteinX + geneY ]_l -c_1-> [ proteinX_geneY ]_l
  r2: [ proteinX_geneY ]_l -c_2-> [ proteinX + geneY ]_l
  r3: [ proteinX_geneY ]_l -c_3-> [ proteinX_geneY + rnaY ]_l
}

NegReg((X,Y),(c_1,c_2),(1)) =
{
  r1: [ proteinX + geneY ]_l -c_1-> [ proteinX_geneY ]_l
  r2: [ proteinX_geneY ]_l -c_2-> [ proteinX + geneY ]_l
}

PostTransc((X),(c_1,c_2,c_3),(1)) =
{
  r1: [ rnaX ]_l -c_1-> [ ]_l
  r2: [ rnaX ]_l -c_2-> [ rnaX + proteinX ]_l
  r3: [ proteinX ]_l -c_3-> [ ]_l
}

NAR((X),(c_1,c_2),(1)) =
{
  NegReg((X,X),(c_1,c_2),(1)) from this
}

IFFL((X,Y,Z),(c_1,c_2,c_3,c_4,c_5,c_6,c_7,c_8),(1)) =
{
In a Synthetic biology scenario modules can describe the molecular interactions involved in a well characterised synthetic construct as a Biobrick that can be reusen in the development of different synthetic cellular designs. For example, the following library illustrates how two different inverters can be designed in an incremental manner and introduced in a library in order to facilitate their inclusion in different synthetic cellular designs:

```
libraryOfModules inverters

PostTransc({X},{c_1,c_2,c_3,c_4,c_5},{l}) =
{ 
  r1: [ rnaX_RNAP ]_l -c_1-> [ rnaX ]_l 
  r2: [ rnaX ]_l -c_2-> [ ]_l 
  r3: [ rnaX ]_l -c_3-> [ rnaX + proteinX_Rib ]_l 
  r4: [ proteinX_Rib ]_l -c_4-> [ proteinX ]_l 
  r5: [ proteinX ]_l -c_5-> [ ]_l 
}

Plac({X},{c_1,c_2,c_3,c_4},{l}) =
{ 
  r1: [ Plac_geneX ]_l -c_1-> [ Plac_geneX + rnaX_RNAP ]_l 
  r2: [ proteinLacI + Plac_geneX ]_l -c_2-> [ Plac_LacI_geneX ]_l 
  r3: [ Plac_LacI_geneX ]_l -c_3-> [ proteinLacI + Plac_geneX ]_l 
  r4: [ IPTG + Plac_LacI_geneX ]_l -c_4-> [ proteinLacI_IPTG + Plac_geneX ]_l 
}

PR({X},{c_1,c_2,c_3,c_4,c_5},{l}) =
{ 
  r1: [ PR_geneX ]_l -c_1-> [ PR_geneX + rnaX_RNAP ]_l 
  r2: [ proteinCI2 + PR_geneX ]_l -c_2-> [ PR_CI2_geneX ]_l 
  r3: [ PR_CI2_geneX ]_l -c_3-> [ proteinCI2 + PR_geneX ]_l 
  r4: [ proteinCI2 + PR_CI2_geneX ]_l -c_4-> [ PR_CI4_geneX ]_l 
  r5: [ PR_CI4_geneX ]_l -c_5-> [ proteinCI2 + PR_CI2_geneX ]_l 
}

Inverter_LacI({X},{},{l}) =
{ 
  PostTransc({LacI},{},{l}) from this 
  Plac({X}{}{l}) from this 
}

Inverter_CI({X},{},{l}) =
{ 
  PostTransc({CI},{},{l}) from this 
  PR({X}{}{l}) from this 
}

endLibraryOfModules
```

The inverters library consists of five modules:
• The **PostTransc module** describes transcription elongation and termination (r1), RNA degradation (r2), translation initiation (r3), translation elongation and termination (r4) and protein degradation (r5). This module can be seen as having the object \textit{rnaX\_RNAP} as input and the object \textit{proteinX} as output. The string-object \textit{rnaX\_RNAP} represents an RNA polymerase that has initiated transcription of the \textit{geneX} and \textit{proteinX} represents the protein product of \textit{geneX}.

![BioBrick representation of our PostTransc module.](image)

• The **Plac module** describes the binding and debinding (r2 and r3) of the repressor \textit{proteinLacI} to and from the lactose operon promoter. This repressor prevents the initiation of the transcription of the gene fused to the promoter represented as the production of the string-object \textit{rnaX\_RNAP} (r1). This module also considers the case when the repressor debinds from the promoter in the presence of the signal \textit{IPTG} (r4). This module can be considered to have the repressor \textit{proteinLacI} and signal \textit{IPTG} as input and \textit{rnaX\_RNAP} as output.

![BioBrick representation of our Plac module.](image)

• The **PR module** describes the cooperative binding and debinding (rules r2 - r5) of the repressor CI to the \textit{PR} promoter of the bacteriophage lambda. This repressor prevents the initiation of the transcription of the gene fused to the promoter represented as the production of the string-object \textit{rnaX\_RNAP} (r1). This module can be considered to have the repressor \textit{proteinCI} as input and \textit{rnaX\_RNAP} as output.

![BioBrick representation of our PR module.](image)

• The **Inverter\_LacI module** uses the repressor \textit{LacI} and the promoter \textit{Plac} to construct a molecular inverter with input transcripts of the \textit{LacI} gene, \textit{rnaLacI\_RNAP}, and output transcripts of the gene fused to the promoter \textit{Plac}, \textit{rnaX\_RNAP}. This is achieved by composing the module \textit{PostTransc} instantiated with \textit{LacI} and its characteristic rates and the module \textit{Plac}.

• The **Inverter\_CI module** uses the repressor \textit{CI} and the promoter \textit{PR} to construct a molecular inverter with input transcripts of the \textit{CI} gene, \textit{rnaCI\_RNAP}, and output transcripts of the gene fused to the promoter \textit{PR}, \textit{rnaX\_RNAP}. This is achieved by composing the module \textit{PostTransc} instantiated with \textit{CI} and its characteristic rates and the module \textit{PR}.

**Specification of Geometric Distribution**

The \textit{spatial distribution} of the different cell types in multi-cellular systems such as tissues or colonies of cells plays a crucial role in processes involved in cell signalling [Burkhard2007]. The \textit{Infobiotics modelling language} allows the
user to capture characteristic spatial distribution in multi-cellular systems using finite point lattices as described in this section.

A finite point lattice, lattice for short, is a grid of regularly distributed spatial points in $\mathbb{R}^n$ ($n=1$ or $2$ in the current version of the Infobiotics Workbench). A lattice is determined by a set of basis vectors $\{b_1,...,b_n\}$, and two sets of lower and upper integer bounds, $\{l_1,...,l_n\}$ and $\{u_1,...,u_n\}$ respectively. The points of a regular lattice are then obtained as all the possible linear combinations of the basis vectors with integer coefficients within the given bounds:

$$\text{Lat} = \{ p = c_1 * b_1 + ... + c_n * b_n : c_i \text{is an integer between} \ l_i \ \text{and} \ u_i \}$$

Note that a point in a lattice is uniquely identified by the coefficients $c_i$ and therefore it will be represented as $(c_1, ..., c_n)$.

Each point in a lattice is associated with a neighbourhood, set of points assumed to be near the given one. A neighbourhood of size $k$ is determined by a set of vectors $\{n_1,...,n_k\}$. Given a point $p$ in a lattice its $k$ neighbours are computed as $p_i = p + n_i$.

We also associate with each point in the lattice a regular polygon, (typically a square, rectangle or hexagon) used to produce a tessellation of the space. This polygon is determined by a set of vectors $\{v_1, ..., v_q\}$ used to compute each vertix, $\text{vertix}_i$, of the polygon associated with a point in the lattice $p$ as $\text{vertix}_i = p + v_i$. The name of the polygon used for the tessellation is normally used as an adjective for the lattice, below we present the most commonly used lattices so far in models developed within the Infobiotics workbench, namely, square, rectangular and hexagonal lattices.

A regular lattice must be specified in the Infobiotics Workbench in a text file with the extension .lat in order to allow its reusability in order multi-cellular systems with the same geometrical distribution but different cell types. The components of a lattice are specified according to the following general skeleton:

```
lattice latticeName
  dimension d
  xmin x1
  xmax x2
  ymin y1
  ymax y2
  parameters
    ...
  endParameters
  basis
    ...
  endBasis
```
Figure 2.3: Example of a *square lattice* with a neighbourhood of four points. The lattice is determined by the basis vectors \( \{ b_1, b_2 \} \), the neighbourhood \( \{ n_1, n_2, n_3, n_4 \} \) and the polygon defined by the vertices \( \{ v_1, v_2, v_3, v_4 \} \).

Figure 2.4: Example of a *rectangular lattice* where each point has a neighbourhood of eight points.
Figure 2.5: Example of a hexagonal lattice with a neighbourhood of six points
An identifier \textit{latticeName} is associated with the lattice specification that consists of the enumeration of the components of a lattice. The dimension, one or two, is declared following the key word \textit{dimension}. The lower and upper bounds for the first and possibly the second coefficients used in the generation of the lattice points is stated next after the keywords \textit{xmin}, \textit{xmax}, \textit{ymin} and \textit{ymax}. The last two can be omitted if the dimension of the lattice is one. Parameters used in the definition of the basis, vertix and neirghbour vectors must be declared within the block \textit{parameters ... endParameters} as \textit{parameter parameterName value=val}.

The basis vectors are specified in the block \textit{basis ... endBasis}, the vertices in the block \textit{vertices ... endVertices} and the neighbours in the block \textit{neighbours ... endNeighbours}. Each vector must be declared as \textit{(firstComponent, secondComponent)}.

For example, in our running example we use the following square lattice

```
lattice rectangular

dimension 2
xmin 0
xmax 10
ymin 0
ymax 30

parameters
  parameter a value = 1
endParameters

basis
  (a,0)
  (0,a)
endBasis

vertices
  (a/2,a/2)
  (-a/2,a/2)
  (-a/2,-a/2)
  (a/2,-a/2)
endVertices

neighbours
  (1,0)
  (-1,0)
  (0,1)
  (0,-1)
endNeighbours
```

\textbf{Specification of Multi-cellular Systems}

Finally, a model of a multi-cellular system with cell types represented by the \textit{SP-systems, SP}_1, \ldots, \textit{SP}_n and spatial distribution captured in a finite point lattice \textit{Lat} is specified as a \textbf{Lattice Population P-system} which distributes many
clones of the different cell types over the corresponding lattice points.

The polygon associated with each point is used to represent the shape of the corresponding cell. The neighbours associated with each point are used when a rule of the form below moving objects to the outside of a compartment is applied in the outermost membrane of a SP-system. In this case one the SP-system located in the neighbour points is chosen randomly and the corresponding objects are placed in its outermost compartment.

An LPP-system enumerates the SP-systems representing the different individual cell types, the finite point lattice describing the geometry of the multi-cellular system and a position function that assigns a SP-system with each point of the lattice.

The model of a multi-cellular system as an LPP-system must be specified in a text file with the extension, .lpp. The different components of the model are specified according to the following format:

```
LPPsystem modelName
   SPsystems
   ...
   endSPsystems
   lattice latticeName from latticeFile
   spatialDistribution
   ...
   endSpatialDistribution
endLPPsystem
```

The model of the multi-cellular system is identified with modelName. In the block SPsystems ... endSPsystems the models of the individual cell types as SP-systems must be enumerated stating the file where they are defined. An individual cell type modelled as a SP-system identified with cellTypeName in file fileName.sps is declared as:

```
SPsystem cellTypeName from fileName.sps
```

Recall that the individual cell types can also be specified in SBML format. In this case the declaration of a cell type is exactly as above except that the extension of the file containing the model must have the extension .sbml.

The finite point lattice capturing the geometry of the multi-cellular system is introduced by specifying an identifier latticeName and the file where it is defined using the keyword from.

Finally, the spatial distribution of the different clones of cell types over the lattice points is specified within the block spatialDistribution ... endSpatialDistribution. Each SP-system identified with SPsystemName is associated with a set of positions occupied by cells of the corresponding type. This is specified using the block positions for SPsystemName ...

```
positions for boundaryCell
   parameters
   ...
   endParameters
   coordinates
   ...
   endCoordinates
endPositions
```

Here the coordinates of the lattice points are declared in the block:

```
coordinates
   x = coordinate1
```
The values coordinate1 and coordinate2 can be specified as mathematical formulas using parameters. These parameters can be defined as taking values within a given range \([ lowerBound \ldots upperBound \) with a specified \( step \) within the block \( \text{parameters} \ldots \text{endParameters} \) as \( \text{parameter parameterName} = lowerBound:step:upperBound \):

\[
\text{parameters} \\
\quad \text{parameter parameterName} = lowerBound:step:upperBound \\
\quad \ldots \\
\text{endParameters}
\]

For example:

LPPsystem pulsePropagation

SPsystems

SPsystem senderCell from senderCell.sps
SPsystem pulsingCell from pulsingCell.sps
SPsystem boundaryCell from pulsingCell.sps
endSPsystems

lattice rectangular from lattice.lat

spatialDistribution

positions for boundaryCell

parameters

\[
\begin{align*}
\text{parameter} \ i & = 0:1:51 \\
\text{parameter} \ j & = 0:11:11
\end{align*}
\]

endParameters
coordinates

\[
\begin{align*}
x & = i \\
y & = j
\end{align*}
\]
endCoordinates
endPositions

positions for boundaryCell

parameters

\[
\begin{align*}
\text{parameter} \ i & = 0:51:51 \\
\text{parameter} \ j & = 0:1:10
\end{align*}
\]

endParameters
coordinates

\[
\begin{align*}
x & = i \\
y & = j
\end{align*}
\]
endCoordinates
endPositions

positions for senderCell

parameters

\[
\begin{align*}
\text{parameter} \ i & = 1:1:5 \\
\text{parameter} \ j & = 1:1:10
\end{align*}
\]

endParameters
coordinates

\[
\begin{align*}
x & = i \\
y & = j
\end{align*}
\]
endCoordinates
endPositions

positions for pulsingCell
parameters
  parameter i=6:1:50
  parameter j=1:1:10
endParameters

coordinates
  x=i
  y=j
endCoordinates

endPositions

endSpatialDistribution
endLPPsystem

Credits:
The Infobiotics modelling language was developed by Francisco J. Romero-Campero with contributions from Jamie Twycross, Jonathan Blakes and Hongqing Cao. It is being used on Systems and Synthetic Biology research projects in the University of Nottingham, U.K.

2.3.2 Multi-compartmental Stochastic Simulations

Introduction

mcss is an application for simulating multi-compartment stochastic P system models. mcss takes a model specified in SBML and simulates it using the multi-compartment Gillespie algorithm. A large number of spatially-distributed compartments containing many chemical species, reactions and transportation channels can be simulated. Templates can be specified which define a set of reactions which can be reproduced in many compartments. mcss is being used to develop Systems/Synthetic Biology computational models of plant systems and bacterial colonies.

Installation

For instructions on how to compile and install mcss, see the README file included with the mcss distribution.

Running mcss

Once installed, mcss is run by typing the following command:: $ mcss PARAMETER_FILE PARAMETER=VALUE PARAMETER=VALUE ...
where PARAMETER_FILE is the filename of an mcss parameter file. Parameters specified on the command line following the filename override the parameters in the parameter file. For example, to run the module1 model provided in the examples directory of the mcss distribution, change to this directory and type:: $ mcss module1.params mcss saves the output of the simulation as a HDF5 data file, the filename of which can be specified in the parameter file. For example, after running the module1 model as described above, the file module1.h5 is created containing the results of the simulation. The mcss-postprocess application, which is included in the mcss distribution, can be used to extract information from the HDF5 output file. Alternatively, the standard HDF5 utilities can be used to examine this file. See http://hdf.ncsa.uiuc.edu/HDF5/ for more information on HDF5.
mcss parameter files

The mcss parameter file is used to provide information to mcss and control various aspects of the simulation. It is in an XML format which has the general form:

```
<parameters>
  <parameterSet name="SimulationParameters">
    <parameter name="PARAMETER NAME" value="PARAMETER VALUE"/>
    <parameter name="PARAMETER NAME" value="PARAMETER VALUE"/>
    ...
  </parameterSet>
</parameters>
```

See the parameter files for the example models for some examples. The allowed parameters are given in the table below.

Model specification

CellDesigner v3.5.2 (http://www.celldesigner.org/) is used to graphically design a model, which is then exported as SBML Level 2. See the example models supplied with the mcss distribution for some examples. Once the model has been designed, export it from CellDesigner by selecting “Export Pure Level 2 Version 1” from the File menu.

Compartment specification

In mcss, compartment names are used to provide information on the templates the compartment defines or uses, and the position of the compartment. This information is given in the SBML name attribute in a number of colon-separated fields. The names of compartments can be changed in CellDesigner by right clicking the compartment and selecting “Change Identity...”.

Compartments must be named as follows:

```
name:t:a,b,...:x,y
```

where,
- name is a string (not necessarily unique) which describes the compartment,
- t is a unique non-negative integer identifying the template the compartment defines,
- a,b,... is a comma-separated list of non-negative integers giving the identifiers of the templates the compartment uses,
- x,y is the position of the compartment, where x and y are non-negative integers.

Some of these field may be empty, depending on the role of the compartment in the model.

For example, all of the following are valid compartment names:

- bacteria:1:
- bacteria::1:0,1
- reaction1:2::0,0
- reaction2:3:2:0,1
- compartment::2,3:1,3

In the first example, “bacteria:1:”, the first colon-separated field indicates that the compartment has the name bacteria. The second field indicates that this compartment defines a template with identifier 1. Compartments which define a template are automatically assumed to use the template they define. The third field is empty, indicating that this
compartment uses no templates other than the one it defines. The fourth field is also empty, indicating that this
compartment is a pure template which defines a set of rules but will not itself be included in the model.

In the second example, “bacteria::1:0,1”, the first field indicates that the compartment has the name bacteria. The
second field is empty, indicating that this compartment does not define any templates. The third field indicates that
the compartment uses the set of rules defined by the template with identifier 1. The fourth field indicates that the
compartment is located at position (0,1).

In the third example, “reaction1:2::0,0”, the first field indicates that the compartment has the name reaction1. The
second field indicates that this compartment defines a template with identifier 2. The third field is empty, indicating
that this compartment uses no templates other than the one it defines. The fourth field indicates that the compartment
is located at position (0,0).

In the fourth example, “reaction2:3:2:0,1”, the first field indicates that the compartment has the name reaction2. The
second field indicates that this compartment defines a template with identifier 3. The third field indicates that, in
addition to the template it defines, this compartment also uses the set of rules defined by the template with identifier
2. The fourth field indicates that the compartment is located at position (0,1).

In the fifth example, “compartment::2,3:1,3”, the first field indicates that the compartment has the name compartment.
The second field is empty, indicating that this compartment does not define any templates. The third field indicates
that the compartment uses the set of rules defined by the templates with identifiers 2 and 3. The fourth field indicates
that the compartment is located at position (1,3).

See the example models included with the mcss distribution for more examples.

**Reaction specification**

A number of different unimolecular and bimolecular reactions can be simulated. See the reaction1 model in the
examples directory for examples of all the reactions that can be simulated.

Reactions whose reactants and products are all in the same compartment must be named as follows:

\[ \text{name} \]

where name is a string (not necessarily unique, may be empty) which identifies the reaction. To specify this name in
CellDesigner, right click the reaction and select “Change Identity...”.

Reactions whose products are in a different compartment to their reactants must be named as follows:

\[ \text{name}:x,y \]

where name is a string (not necessarily unique, may be empty) which identifies the reaction, and x,y is a vector
specifying the offset to compartment the products are to be placed in, where x and y are integers. For example, if a
reactions named re1:1,0 is defined in a compartment with position (1,3), then the reaction will place its products in the
compartment at position (1,3)+(1,0)=(2,3) i.e. the compartment on its right. If the reaction was named re1:0,-1 then
its products would be placed in the compartment at position (1,3)+(0,-1)=(1,2) i.e. the compartment above.

A reaction constant must also be specified for each reaction. To specify this constant in CellDesigner, right click on
the reaction and select “Edit Reaction...”. Now click the KineticLaw Edit button. Due to a bug in libSBML, the “math”
box at the top must contain something, so enter the id of the reaction constant. Click on the New button to create a
new parameter, and enter the id (arbitrary) of the reaction constant, for example “c1”, and a value for this constant.
Only create one parameter for each reaction.

Reaction constants can also be sampled from distributions. Create the constant as described above, entering the id of
the reaction constant, and in the same window, change the “constant” option from true to false. The distribution type
and parameters are specified in the “name” box. Distribution-based reaction constants must be named as follows:
where type is a string describing the distribution to be used, mean is the distribution mean, and sd is the distribution standard deviation. The following strings are valid for the type attribute: gaussian (Gaussian distribution). For example, the name gaussian:0.3:0.1 indicates that the value of the reaction constant will be sampled from a Gaussian distribution with mean 0.3 and standard deviation 0.1. Negative reaction constant values are set to zero.

Species specification

Species can be named arbitrarily. To set the initial amount of molecules present for a species, in CellDesigner right click on the species and select “Edit Species...”, where you will see a box where you can enter the initial amount. By default, initial amounts are only set for species in the template compartment, although you can set the duplicate_initial_amounts parameter in the parameter file to 1 to reproduce the initial amounts in all compartments which use this template. If you want the amount of a species to be constant then you can select the constant option in the “Edit Species...” dialogue in CellDesigner. The level of this species will then always stay at its initial amount, even if the species is involved in any reactions.

License

The mcss distribution, including all source code, model examples, and documentation, are the copyright of Jamie Twycross, and released under the GNU GPL version 3 license.

Credits

mcss was written by Jamie Twycross, with contributions from Francisco Romero-Campero, Jonathan Blakes and James Smaldon. It is being used on Systems Biology research projects in the Centre for Plant Integrative Biology and the School of Computer Science, University of Nottingham, U.K. This work is funded by grants from the BBSRC grant BB/D0196131.

For further information or any questions please contact jpt AT cpib.ac.uk.

copyright 2008, 2009 Jamie Twycross, released under GNU GPL version 3.

2.3.3 Model Formal Analysis using Model Checking

Introduction

pmodelchecker is an application that facilitates the use of formal model analysis using Model Checking of spatio-temporal properties of P system models developed within the Infobiotics workbench. pmodelchecker receives as input a model developed as specified in section and a list of temporal logic formulas formalising some spatio-temporal properties to be checked against the dynamics of the model.

pmodelchecker uses two different stochastic model checkers, PRISM and MC2. When using PRISM it generates a model in the reactive modules language needed in PRISM in order to check the input logic formulas. When using MC2 it generates the needed simulation samples by running mcss, the multicomparmental simulator introduced in the previous section.
Installlation

For instructions on how to compile and install `pmodelchecker`, see the README file included with the `pmodelchecker` distribution.

Running `pmodelchecker`

In order to run `pmodelchecker` after its installation run the following command:

```
$ pmodelchecker PARAMETER_FILE
```

where `PARAMETER_FILE` is an xml file declaring the input parameters required to perform model checking using one of the two stochastic model checkers integrated in Infobiotics, PRISM or MC2. For example, in order to run the examples in the directory `PRISMexamples/` browse to this directory provided in the directory `examples/` of the `pmodelchecker` distribution and type one of the commands below depending on which example you want to run:

```
$ pmodelchecker NAR1.params
$ pmodelchecker NAR2.params
```

In a similar manner to run the examples for MC2 browse to the directory `MC2examples/` inside the directory `examples/` provided in the `pmodelchecker` distribution and type the command below:

```
$ pmodelchecker NAR_MC2.params
```

The output, probabilities or expected values for temporal logic formulas, is produced into the result file specified as in the parameter file as explained below.

`pmodelchecker` parameter files

The `pmodelchecker` parameter file provides information to determine which specific model checker to use, PRISM or MC2, and some parameters to control the application of these model checkers like the use of verification versus approximation or the number of samples to consider in the case of a simulative or approximative approach.

The parameter file is in an XML format which has the general form:

```
<parameters>
  <parameterSet name="SimulationParameters">
    <parameter name="PARAMETER NAME" value="PARAMETER VALUE"/>
    <parameter name="PARAMETER NAME" value="PARAMETER VALUE"/>
    ...
  </parameterSet>
</parameters>
```

The specific parameters are given in the table below:
<table>
<thead>
<tr>
<th>PARAMETER NAME</th>
<th>DESCRIPTION</th>
<th>VALUE</th>
<th>RESTRICTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_specification</td>
<td>Name of the file containing the model specification as an LPP-system</td>
<td>String</td>
<td>None</td>
</tr>
<tr>
<td>temporal_formulas</td>
<td>Name of the file containing the temporal logic formulas formalising the properties to check</td>
<td>String</td>
<td>None</td>
</tr>
<tr>
<td>model_checker</td>
<td>Name of the model checker to use PRISM or MC2</td>
<td>String</td>
<td>None</td>
</tr>
<tr>
<td>PRISM_model</td>
<td>Name of the file where to output the translation of our model into the PRISM language</td>
<td>String</td>
<td>Only when using PRISM</td>
</tr>
<tr>
<td>task</td>
<td>Task to perform when using PRISM as model checker. Translate LPP-system into the PRISM language, build the corresponding Markov chain, verify or approximate the input properties</td>
<td>Translate/Build/Verify/Approximate</td>
<td>Only when using PRISM</td>
</tr>
<tr>
<td>model_parameters</td>
<td>Stating the values of the parameters in the model as follows param=lb:ub:s,param=lb:ub:s, ... where lb is the lower bound, up is the upper bound and s is the step</td>
<td>String</td>
<td>Only when using PRISM</td>
</tr>
<tr>
<td>formula_parameters</td>
<td>A string stating the values of the parameters in the formulas as follows param=lb:ub:s,param=lb:ub:s, ... where lb is the lower bound, up is the upper bound and s is the step</td>
<td>String</td>
<td>Only when using PRISM</td>
</tr>
<tr>
<td>states_file</td>
<td>Name of the file where to output the states of the Markov chain generated from the LPP-system</td>
<td>String</td>
<td>Only when using PRISM with task Build or Verify</td>
</tr>
<tr>
<td>transitions_file</td>
<td>Name of the file where to output the transitions of the Markov chain generated from the LPP-system</td>
<td>String</td>
<td>Only when using PRISM with task Build or Verify</td>
</tr>
<tr>
<td>number_samples</td>
<td>Number of simulations to generate when taking an approximate or simulative approach to model checking</td>
<td>Integer</td>
<td>Only when using PRISM with task Approximate or whenever using MC2</td>
</tr>
<tr>
<td>precision</td>
<td>Precision to achieve with respect to real value when generating an estimate using approximate or simulative model checking</td>
<td>Double</td>
<td>Only when using PRISM with task Approximate</td>
</tr>
<tr>
<td>confidence</td>
<td>Confidence to achieve with respect to real value when generating an estimate using approximate or simulative model checking</td>
<td>Double</td>
<td>Only when using PRISM with task Approximate</td>
</tr>
<tr>
<td>pathMC2</td>
<td>Location of the executable file for MC2</td>
<td>String</td>
<td>Only when using MC2</td>
</tr>
<tr>
<td>simulations_generatedHDF5</td>
<td>Flag to determine if the simulations needed to run MC2 are available in HDF5 format</td>
<td>true/false</td>
<td>Only when using MC2</td>
</tr>
<tr>
<td>simulations_generatedMC2</td>
<td>Flag to determine if the simulations needed to run MC2 are available in MC2 format</td>
<td>true/false</td>
<td>Only when using MC2</td>
</tr>
<tr>
<td>simulations_file_hdf5</td>
<td>Name of the file containing the simulations in HDF5 format or where to write them when using mcss</td>
<td>String</td>
<td>Only when using MC2 and the flag simulations_generatedMC2 = false</td>
</tr>
<tr>
<td>simulations_file_MC2</td>
<td>Name of the file containing the simulations in MC2 format or where to write them</td>
<td>String</td>
<td>Only when using MC2</td>
</tr>
<tr>
<td>mcss_params_file</td>
<td>Name of the file containing the parameters to run mcss in order to generate the necessary simulations</td>
<td>String</td>
<td>Only when using MC2 and the flag simulations_generatedHDF5 = false and simulations_generatedMC=false</td>
</tr>
<tr>
<td>results_file</td>
<td>Name of the file where to write the answers to the temporal logic formulas generated by the model checker</td>
<td>String</td>
<td>None</td>
</tr>
</tbody>
</table>
License

The *pmodelchecker* distribution, including all source code, model examples, and documentation, are the copyright of the Infobiotics Team (Hongqing Cao, Claudio Lima, Natalio Krasnogor, Francisco Romero-Campero, Jamie Twycross, and Jonathan Blakes) and is released under the GNU GPL version 3 license.

Credits

*pmodelchecker* was written by Francisco J. Romero-Campero and is being used on Systems/Synthetic Biology research projects in the University of Nottingham, U.K.

For further information or any questions please contact fxc AT cs.nott.ac.uk.

*copyright 2009 Infobiotics Team, released under GNU GPL version 3.*

2.3.4 Structure and Parameter Optimization with *poptimizer*

*poptimizer* Documentation

Introduction

*poptimizer* is an application for optimizing the structure and parameters of stochastic P system models using evolutionary algorithms. *poptimizer* takes a library of modules that represent basic biological processes of interest and combines them in many different ways to discover a possible assembly that mimics the behavior of the target data. During the search process, each model is evaluated by simulating its behavior with *mcss*. *poptimizer* and *mcss* are being used to develop Systems and Synthetic Biology computational models of bacterial colonies and plant systems.

Installation

For instructions on how to compile and install *poptimizer*, see the README file included with the *poptimizer* distribution.

Running *poptimizer*

After installed, *poptimizer* is run by typing the following command:

```
$ poptimizer PARAMETER_FILE
```

where *PARAMETER_FILE* is a file containing the input parameters required by *poptimizer*. For example, to run the promoter model optimization provided in the directory *examples/* of the *poptimizer* distribution, change to the corresponding directory and type:

```
$ poptimizer all_para_promoter_inputpara.xml
```

The output of the optimization procedure can be inspected in several files. The log information with the generation number, number of function evaluations, and fitness of the best solution is saved to file *evolveprocess_Run0.txt*. The best P system obtained at the end of the optimization is saved to *bestPsystem_Run0.txt* and the corresponding time series to *bestsimulation_Run0_initfile0.txt*.

Currently, *poptimizer* can process two types of input data from cell systems biological data:
1. time series data of multiple target objects under one initial state.
2. time series data of multiple target objects under different initial states.

**poptimizer Parameter File**

The structure of the parameter file required for executing *poptimizer* is described in file *poptimizer-parameters-template.xml* under directory *src/poptimizer/*. Different examples for the parameter file can be seen under directory *examples/*.

**Models**

The models built by *poptimizer* have flexible structure and parameters. A particular model is composed by a set of elementary modules (previously specified in a library) that act as the ‘building blocks’. The user can define his own module library based on specific knowledge or simply on elementar biological motifs described in Systems Biology literature.

While certain modules can have fixed rules and kinetic constants (fixed module library), others can be instantiated with different objects (proteins, genes, etc) and parameter values (non-fixed library). Many kinetic constants referring to well-known reactions can be taken from the literature and introduced in the library, where others need to be evolved by the parameter optimization methods available in *poptimizer*.

**Model Structure Optimization**

The optimization of the model structure concerns with the choice of which modules should compose the model. The number of modules and their corresponding instantiation (according to a choice of different objects) is also explored to minimize the error between the output data generated by the model and the target data. A genetic algorithm that selects, recombines, and mutates different sets of modules is used to optimize the model structure.

**Model Parameters Optimization**

The optimization of the model parameters concerns with learning the appropriate kinetic constants corresponding to each one of the rules specified in the modules. When the kinetic constants are not known from literature, the module library specifies the parameter ranges (and a choice of linear/logarithmic scale) for each kinetic constant. The parameter optimization methods currently available include genetic algorithms (GA), differential evolution (DE), opposition differential evolution (ODE), and the covariance matrix adaptation evolution strategy (CMA-ES).

**Fitness Function**

*poptimizer* can use two different fitness functions to quantify the quality of candidate models. These are:

1. **Equal Weighted Sum**: The fitness is given by the arithmetic sum of the RMSE (between target and model data) for each one of the time series. This is a common method for calculating the total error of several time series with similar magnitude.
2. **Random Weighted Sum**: The fitness is obtained by a weighted sum of the errors that is adjusted according to a normalized weight vector randomly generated. A high number of different weight vectors is generated and used in the fitness calculation to average out the randomness given to the weights. This method allows a more wide exploration when fitting a model to time series of different orders of magnitude.
Additional Information

More detailed information about the methodology can be found in the paper entitled *Evolving Cell Models for Systems and Synthetic Biology*, to appear in the Systems and Synthetic Biology journal.

Examples

This section briefly describes three different running examples for *poptimizer*. The first two examples are taken from the reference paper cited above and the third refers to a pulse generator with different initial conditions.

threegene

This case study investigates regulatory networks consisting of three genes that are able to produce a pulse in the expression of a specific gene. The corresponding files can be found in *examples/threegene/*. To run this example, change to the corresponding directory, and type:

```
$ poptimizer threegene_inputpara.xml
```

The non-fixed module library used is specified in file *threegene_module_library.xml*, the target data in *target_data_threegene.txt*, and the initial values for each of the genes in *initial_values_threegene.txt*.

promoter

This case study investigates a gene regulatory network consisting of five genes that is able to behave as a bandwidth detector. The corresponding files can be found in *examples/promoter/*. To run this example, change to the corresponding directory, and type:

```
$ poptimizer all_para_promoter_inputpara.xml
```

The non-fixed module library used is specified in file *all_para_module_library_promoter.xml*, the target data in *target_data_promoter.txt*, and the initial values for each of the genes in *initial_values_promoter.txt*.

fourinitial

The last example deals with a network of at most five genes to simulate a pulse generator for one the genes under different initial conditions. The corresponding files can be found in *examples/fourinitial/*. To run this example, change to the corresponding directory, and type:

```
$ poptimizer four_initial_inputpara.xml
```

A fixed module library specified in file *library2.xml* is now used together with the non-fixed library *library1-lin.xml*. The target data is now specified in four different files (*target1.txt*, *target2.txt*, *target3.txt*, *target4.txt*), as well as the initial values (*initials1.txt*, *initials2.txt*, *initials3.txt*, *initials4.txt*).
**poptimizer Software**

**License**

The *poptimizer* distribution, including all source code, model examples, and documentation, are the copyright of the Infobiotics Team (Hongqing Cao, Claudio Lima, Natalio Krasnogor, Francisco Romero-Campero, Jamie Twycross, and Jonathan Blakes) and is released under the GNU GPL version 3 license.

**Credits**

*poptimizer* was written by Hongqing Cao, with contributions from Claudio Lima, Natalio Krasnogor, Jamie Twycross, Francisco Romero-Campero, and Jonathan Blakes. It is being used on Systems Biology research projects in the Centre for Plant Integrative Biology and the School of Computer Science, University of Nottingham, U.K. This work is funded by grants from the BBSRC grant BB/D0196131.

For further information or any questions please contact cvf AT cs.nott.ac.uk.

*copyright 2009 Infobiotics Team, released under GNU GPL version 3.*
3.1 The Repressilator

3.1.1 Introduction

The Repressilator is one the first implemented synthetic genetic circuits. It was developed by Michael B. Elowitz and Stanislas Leibler [Elowitz2000]. It has been used as a canonical example in synthetic biology in various P system studies [Gheorghe2009].

The Repressilator consists of three genes codifying three repressors. Namely, the operon lactose repressor, lacI, the repressor form the tetracycline transposon, tetR, and a repressor from the λ phage virus, cI. These genes are connected in a synthetic gene regulatory network such that LacI represses the expression of the tetR gene, which in turn represses the cI gene. Finally, the cycle is closed as CI represses the expression of the lacI gene.

The complete model of the repressilator developed using Infobiotics workbench is available from this link.

3.1.2 The Model

Our model of the Repressilator consists of a single bacterial cell modelled as a SP-system identified with the name repressilatorCell. A bacterial colony of this cell type is then created by distributed cellular clones over the points of a rectangular lattice.

The molecular interactions in the corresponding SP-system are defined in a modular manner using the modules from the library basicLibrary below:
libraryOfModules basicLibrary

# A module representing the unregulated expression of a gene X #
# UnReg({X},{c_1, c_2, c_3, c_4},{l}) = 
{ rules: 
  # Transcription of geneX #
  r1: [ geneX ]_l -c_1-> [ geneX + rnaX ]_l
  # Degradation of the RNA #
  r2: [ rnaX ]_l -c_2-> [ ]_l
  # Translation of the RNA #
  r3: [ rnaX ]_l -c_3-> [ rnaX + proteinX ]_l
  # Degradation of the protein #
  r4: [ proteinX ]_l -c_4-> [ ]_l
}

# A module representing the positive regulation of a protein X #
# over a gene Y #
PosReg({X,Y},{c_1, c_2, c_3, c_4, c_5, c_6},{l}) = 
{ rules: 
  # Binding and debinding of the transcription factor proteinX to geneY #
  r1: [ proteinX + geneY ]_l -c_1-> [ proteinX_geneY ]_l
  r2: [ proteinX_geneY ]_l -c_2-> [ proteinX + geneY ]_l
  # Transcription of geneY when proteinX is bound to its promoter #
  r3: [ proteinX_geneY ]_l -c_3-> [ proteinX_geneY + rnaY ]_l
  # Degradation of the RNA #
  r4: [ rnaY ]_l -c_4-> [ ]_l
  # Translation of the RNA #
  r5: [ rnaY ]_l -c_5-> [ rnaY + proteinY ]_l
  # Degradation of the protein #
  r6: [ proteinY ]_l -c_6-> [ ]_l
}

# A module representing the negative regulation of a protein X #
# over a gene Y #
NegReg({X,Y},{c_1, c_2},{l}) = 
{ rules: 
  # Binding and debinding of the transcription factor proteinX to gene Y #
  r1: [ proteinX + geneY ]_l -c_1-> [ proteinX_geneY ]_l
  r2: [ proteinX_geneY ]_l -c_2-> [ proteinX + geneY ]_l
}

# A module representing the cooperative regulation of a protein# # X over a gene Y #
CoopNegReg({X,Y},{c_1, c_2, c_3, c_4, c_5, c_6},{l}) = 
{ rules: 
  r1: [ proteinX + geneY ]_l -c_1-> [ proteinX_geneY ]_l
  r2: [ proteinX_geneY ]_l -c_2-> [ proteinX + geneY ]_l
  r3: [ proteinX + proteinX_geneY ]_l -c_3-> [ proteinX2_geneY ]_l
  r4: [ proteinX2_geneY ]_l -c_4-> [ proteinX + proteinX_geneY ]_l
  r5: [ proteinX_geneY ]_l -c_5-> [ proteinX_geneY + rnaY ]_l
  r6: [ proteinX2_geneY ]_l -c_6-> [ proteinX2_geneY + rnaY ]_l
}
endLibraryOfModules
The cell type or bacterial strain carrying the repressilator is modelled using the following SP-system:

```plaintext
# Author: Francisco J. Romero-Campero #
# Date: 14 May 2010 #
# Description: A single cell carrying the repressilator #
SPsystem repressilatorCell

# Molecular species present in the system #
alphabet
  geneCI
  geneLacI
  geneTetR
  proteinCI
  proteinCI2_geneLacI
  proteinCI_geneLacI
  proteinLacI
  proteinLacI2_geneTetR
  proteinLacI_geneTetR
  proteinTetR
  proteinTetR2_geneCI
  proteinTetR_geneCI
  rnaCI
  rnaLacI
  rnaTetR
endAlphabet

# This model consists of a single compartment #
compartments
  bacterium
endCompartments

# In the initial state of the system only a single copy of the #
# genes lacI, ci and tetR are present #
initialMultisets
  initialMultiset bacterium
    geneLacI 1
    geneCI 1
    geneTetR 1
endInitialMultiset
endInitialMultisets

# The molecular interactions involved in the repressilator #
rules
  ruleSet bacterium
    # CI represses cooperatively the lacI gene which expressed constitutively otherwise #
    CoopNegReg({CI,LacI},{1,224,1,9,0.0005,0.0005},{bacterium}) from basicLibrary.plb
    UnReg({LacI},{0.5,0.00578,0.167,0.00116},{bacterium}) from basicLibrary.plb
    
    # LacI represses cooperatively the tetR gene which expressed constitutively otherwise #
    CoopNegReg({LacI,TetR},{1,224,1,9,0.0005,0.0005},{bacterium}) from basicLibrary.plb
    UnReg({TetR},{0.5,0.00578,0.167,0.00116},{bacterium}) from basicLibrary.plb
    
    # TetR represses cooperatively the ci gene which expressed constitutively otherwise #
    CoopNegReg({TetR,CI},{1,224,1,9,0.0005,0.0005},{bacterium}) from basicLibrary.plb
    UnReg({CI},{0.5,0.00578,0.167,0.00116},{bact}) from basicLibrary.plb
endRuleSet
```

3.1. The Repressilator
The geometry of a bacterial colony of the cell type or bacterial strain represented in the previous model is captured using the following rectangular lattice:

```
# Author: Francisco J. Romero-Campero  #
# Date: July 2010                     #
# Description: A rectangular lattice of size 5x5 #

lattice rectangularLattice

    # Dimension of the lattice and lower/upper bounds #
    dimension 2
    xmin 0
    xmax 4
    ymin 0
    ymax 4

    # Parameters used in the definition of the rest of components defining the lattice #
    parameters
        parameter b1 value = 2
        parameter b2 value = 1
    endParameters

    # Basis vector determining the points in the lattice #
    # in this case we have a rectangular lattice      #
    basis
        (b1,0)
        (0,b2)
    endBasis

    # Vertices used to determine the shape of the outmost membrane #
    # of the SP systems located on each point of the lattice    #
    vertices
        (b1/2,b2/2)
        (-b1/2,b2/2)
        (-b1/2,-b2/2)
        (b1/2,-b2/2)
    endVertices

    # Vectors pointing at the neighbours of each point of the lattice #
    neighbours
        (1,0)  (1,1)  (0,1)  (-1,1)
        (-1,0) (-1,-1) (0,-1) (1,-1)
    endNeighbours

endLattice
```

Finally, the model of a bacterial colony is obtained by distributing cellular clones of the bacterial cell carrying the repressilator over the points of the previous lattice. This is modelled using the LPP-system below:

```
# Author: Francisco J. Romero-Campero  #
# Date: July 2010                     #
# Description: A multicellular system consisting of a bacterial colony #
# carrying the repressilator #
```
LPPsystem repressilatorColony

# Cell types specified as individual SP systems #
SPsystems
  SPsystem repressilator from repressilator.sps
endSPsystems

# The geometry of the system is determined using a regular finite point lattice #
lattice rectangular from rectangular.lat

# Spatial distribution of the cells over the lattice #
spatialDistribution

  # Bacteria carrying the repressilator are distributed over all the points of the lattice #
  positions for repressilator
    parameters
      parameter i = 0:1:4
      parameter j = 0:1:4
    endParameters
    coordinates
      x = i
      y = j
    endCoordinates
  endPositions
endSpatialDistribution
endLPPsystem

The complete model of the repressilator developed using the Infobiotics workbench can be downloaded from this link.

### 3.1.3 Simulations

Stochastic simulations of our model of the repressilator can be run using the Infobiotics workbench. For this, please load using the provided interface the simulation parameter file, `simulation_parameters.params`, provided with the files comprising this example. Be patient; these simulations could take a few minutes.

Below we show the evolution over time of the number of proteins LacI, CI and TetR in three different bacteria from the colony. Note that the system exhibits oscillatory behaviour that is not synchronised between different bacteria.

The oscillatory behaviour of the system and the lack of synchronisation between bacteria can be observed more clearly in the dynamics of the entire colony:

This video shows the spatio-temporal evolution of the number of proteins LacI, CI and TetR in our model of a bacterial colony carrying the repressilator.

### 3.1.4 Model Checking

Model checking of some stochastic properties of the repressilator were run using the Infobiotics workbench. For this, please load using the provided interface the model checking parameter file, `model_checking_mc2.params`, provided with the files comprising this example. Be patient; this analysis could take a few minutes.

Below we show the analysed properties. They are used to compute the probability of having more or fewer than 300 proteins of LacI, CI and TetR simultaneously over different time points of the evolution.
Chapter 3. Model Repository
3.1. The Repressilator

![Graph showing proteinCI, proteinLacI, and proteinTetR](image)

Molecules of proteinCI at 37°C: 7,247 molecules
Molecules of proteinLacI at 37°C: 7,553 molecules
Molecules of proteinTetR at 37°C: 6,083 molecules

![Heatmaps showing distribution](image)
Below we present the results for the properties above. In both cases the probability is zero which suggests that the three proteins simultaneously cannot be above or below 300 molecules, they should be alternatively oscillating.

### 3.2 Pulse Generator

#### 3.2.1 Introduction

The pulse generator example consists of the synthetic bacterial colony designed by Ron Weiss' group in [Basu2005]. This model implements the propagation of a wave of gene expression in a bacterial colony. The complete model can be downloaded from this [link](#).

The pulse generator consists of two different bacterial strains, sender cells and pulsing cells:

- **Sender cells** contain the gene *luxI* from *Vibrio fischeri*. This gene codifies the enzyme LuxI responsible for the synthesis of the molecular signal 3OC6-HSL (AHL). The *luxI* gene is expressed constitutively under the regulation of the promoter PLtetO1 from the tetracycline resistance transposon.

- **Pulsing cells** contain the *luxR* gene from *Vibrio fischeri* that codifies the 3OC6-HSL receptor protein LuxR. This gene is under the constitutive expression of the promoter PluxL from *Vibrio fischeri*. It also contains the gene *ci* from lambda phage codifying the repressor CI under the regulation of the promoter PluxR that is activated upon binding of the transcription factor LuxR-3OC6_2. Finally, this bacterial strain carries the gene *GFP* that codifies the green fluorescent protein under the regulation of the synthetic promoter PluxPR combining the Plux promoter (activated by the transcription factor LuxR-3OC6_2) and the PR promoter from lambda phage (repressed by the transcription factor CI).

The bacterial strains above are distributed in a specific spatial distribution. Sender cells are located at one end of the bacterial colony and the rest of the system is filled with pulsing cells as the figure below shows:

#### 3.2.2 The Model

Our model of the Pulse Generator uses a module library describing the regulation of the different promoters used in the two bacterial strains. This library is presented below:
3.2. Pulse Generator
# This module represents the constitutive expression of a gene X under the regulation of the promoter PLtetO1 from the tetracycline resistance transposon PLtetO1((X),(c_1),(l)) =

```
# This module represents a promoter
```

type: promoter

```
# DNA sequence corresponding to biobrick BBa_R0040 from the Registry of Standard Biological Parts
sequence: TCCCTATCAGTGATAGAGATTGACATCCCTATCAGTGATAGAGATACTGAGCAC
```

```
rules:
  r1: [ PLtetO1_geneX ]_l -c_1-> [ PLtetO1_geneX + rnaX_RNAP ]_l
```

# This module represents the constitutive expression of a gene X under the regulation of the promoter PluxL from Vibrio fischeri PluxL((X),(c_1),(l)) =

```
# This module represents a promoter
```

type: promoter

```
# DNA sequence corresponding to biobrick BBa_R0063 from the Registry of Standard Biological Parts
sequence: ACCTGTACGATCCTACAGGTGCTTATGTTAAGTAATTGTATTCCCCTATCAGTAGAGATCTACTGACCC
```

```
rules:
  r1: [ PluxL_geneX ]_l -c_1-> [ PluxL_geneX + rnaX_RNAP ]_l
```

# This module represents the positively regulated expression of a gene X under the control
# of the promoter PluxR which is activated by LuxR_2
PluxR((X),(c_1, c_2, c_3),(1)) =
{
    # This module represents a promoter#
    type: promoter

    # DNA sequence corresponding to biobrick BBa_R0062 from the Registry of Standard Biological Parts#
    sequence: ACCTGTAGGATCGTACAGGTTTACGCAAGAAAATGGTTTGTTATAGTCGAATTTA

    rules:
    r1: [ LuxR2 + PluxR_geneX ]_l -c_1-> [ PluxR_LuxR2_geneX ]_l
    r2: [ PluxR_LuxR2_geneX ]_l -c_2-> [ LuxR2 + PluxR_geneX ]_l
    r3: [ PluxR_LuxR2_geneX ]_l -c_3-> [ PluxR_LuxR2_geneX + rnaX_RNAP ]_l
}

# This module represents the positive/negative regulated expression of a gene X under the control of the synthetic promoter PluxPR that combines the activation by LuxR2 with the repression by CI2
PluxPR((X),(c_1,c_2,c_3,c_4,c_5,c_6,c_7,c_8,c_9),(1)) =
{
    # This module represents a promoter#
    type: promoter

    # DNA sequence corresponding to biobrick BBa_I1051 from the Registry of Standard Biological Parts#
    sequence: ACCTGTAGGATCGTACAGGTTTACGCAAGAAAATGGTTTGTTATAGTCGAAATACCTCTGCGGTTGATA

    rules:
    r1: [ LuxR2 + PluxPR_geneX ]_l -c_1-> [ PluxPR_LuxR2_geneX ]_l
    r2: [ PluxPR_LuxR2_geneX ]_l -c_2-> [ LuxR2 + PluxPR_geneX ]_l
    r3: [ LuxR2 + PluxPR_CI2_geneX ]_l -c_3-> [ PluxPR_LuxR2_CI2_geneX ]_l
    r4: [ PluxPR_LuxR2_CI2_geneX ]_l -c_4-> [ LuxR2 + PluxPR_CI2_geneX ]_l
    r5: [ CI2 + PluxPR_geneX ]_l -c_5-> [ PluxPR_CI2_geneX ]_l
    r6: [ PluxPR_CI2_geneX ]_l -c_6-> [ CI2 + PluxPR_geneX ]_l
    r7: [ CI2 + PluxPR_LuxR2_geneX ]_l -c_7-> [ PluxPR_LuxR2_CI2_geneX ]_l
    r8: [ PluxPR_LuxR2_CI2_geneX ]_l -c_8-> [ CI2 + PluxPR_LuxR2_geneX ]_l
    r9: [ PluxPR_LuxR2_geneX ]_l -c_9-> [ PluxPR_LuxR2_geneX + rnaX_RNAP ]_l
}

endLibraryOfModules

An additional module library describing several post-transcriptional regulatory mechanisms is also used in our model:

# Author: Francisco J. Romero-Campero #
# Date: February 2010 #
# Description: This module library describes some general post-transcriptional regulatory mechanisms#
libraryOfModules postTrasncriptionalRegulation

# This module represents transcription termination, translation, rna and protein degradation#
PostTransc((X),(c_1,c_2,c_3,c_4,c_5),(1)) =
{
    rules:
    r1: [ rnaX_RNAP ]_l -c_1-> [ rnaX ]_l
    r2: [ rnaX ]_l -c_2-> [ rnaX + proteinX_Rib ]_l
    r3: [ rnaX ]_l -c_3-> [ ]_l
    r4: [ proteinX_Rib ]_l -c_4-> [ proteinX ]_l
    r5: [ proteinX ]_l -c_5-> [ ]_l
}

3.2. Pulse Generator 89
Infobiotics Workbench, Release 0.0.1

# This module represents the dimerisation of a protein X #
Dim((X,Y),(c_1,c_2),(1)) =
{
    rules:
    r1: [ proteinX + proteinX ]_l -c_1-> [ Y ]_l
    r2: [ Y ]_l -c_2-> [ ]_l
}

# This module represents the dimerisation of a protein X in the presence of a signal S #
DimSig((X,S,Y),(c_1,c_2,c_3,c_4),(1)) =
{
    rules:
    r1: [ proteinX + signalS ]_l -c_1-> [ proteinX_S ]_l
    r2: [ proteinX_S ]_l -c_2-> [ ]_l
    r3: [ proteinX_S + proteinX_S ]_l -c_3-> [ Y ]_l
    r4: [ Y ]_l -c_4-> [ ]_l
}

# This module represents the free diffusion of a signal X in a rectangular lattice #
Diffusion((X),(c_1),(1)) =
{
    rules:
    r1: [ signalX ]_l = (1,0) = [ ] -c_1-> [ ]_l = (1,0) = [ signalX ]
    r2: [ signalX ]_l = (-1,0) = [ ] -c_1-> [ ]_l = (-1,0) = [ signalX ]
    r3: [ signalX ]_l = (0,1) = [ ] -c_1-> [ ]_l = (0,1) = [ signalX ]
    r4: [ signalX ]_l = (0,-1) = [ ] -c_1-> [ ]_l = (0,-1) = [ signalX ]
}

endLibraryOfModules

The bacterial strain, senderCell, producing the signal 3OC6-HSL (AHL) is modelled using the SP-system model below:

# Author: Francisco J. Romero-Campero #
# Date: February 2010 #
# Description: This SP-system models the sender cell strain developed by Ron Weiss group #
SPsystem senderCell

# Molecular species present in the system #
alphabet
    PLtetO1_geneLuxI
    proteinLuxI
    proteinLuxI_Rib
    rnaLuxI
    rnaLuxI_RNAP
    signal3OC6
endAlphabet

# Compartments of the system #
compartments
    bacterium
endCompartments

# Initial number of molecules in the compartments #
initialMultisets
    initialMultiset bacterium
        PLtetO1_geneLuxI 1
endInitialMultiset
endInitialMultisets

# Molecular interactions produced by the synthetic circuit #
ruleSets

ruleSet bacterium

    # Transcriptional fusion between the PLtetO1 and geneLuxI gene #
    PLtetO1({LuxI},{0.1},{bacterium}) from promoterLibrary.plb
    # Post-transcriptional processes associated with LuxI #
    PostTransc({LuxI},{3.36,0.0667,0.004,3.78,0.0667},{bacterium}) from postTranscriptionalLibrary.plb
    # Signal synthesis #
    r1: [ proteinLuxI ]_bacterium -c1-> [ proteinLuxI + signal3OC12 ]_bacterium
        # Signal diffusion #
    Diffusion([3OC6],[2],[bacterium]) from postTranscriptionalLibrary.plb

endRuleSet

endRuleSets

endSPsystem

The bacterial strain, pulsingCell, producing a pulse of GFP protein as a response to the signal 3OC6-HSL (AHL) is modelled using the SP-system model below:

# Author: Francisco J. Romero-Campero #
# Date: February 2010 #
# Description: This SP-system models the pulsing cell strain developed by Ron Weiss’ group #
SPsystem pulsingCell

    # Molecular species present in the system #
alphabet
    CI2
    LuxR2
    PluxL_geneLuxR
    PluxPR_CI2_geneGFP
    PluxPR_LuxR2_CI2geneGFP
    PluxPR_LuxR2_geneGFP
    PluxPR_geneGFP
    PluxR_LuxR2_geneCI
    PluxR_geneCI
    proteinCI
    proteinCI_Rib
    proteinGFP
    proteinGFP_Rib
    proteinLuxR
    proteinLuxR_3OC6
    proteinLuxR_Rib
    rnaCI
    rnaCI_RNAP
    rnaGFP
    rnaGFP_RNAP
    rnaLuxR
    rnaLuxR_RNAP
    signal3OC6
endAlphabet

3.2. Pulse Generator
# Compartments in the system#
compartments
  bacterium
endCompartments

# Initial number of molecules present in the system#
initialMultisets
  initialMultiset bacterium
    PluxL_geneLuxR 1
    PluxR_geneCI 1
    PluxPR_geneGFP 1
  endInitialMultiset
endInitialMultisets

# Molecular interactions produced by the synthetic gene circuit#
rulesets
  ruleSet bacterium
    # Transcriptional fusion between the PluxL and the LuxR gene#
    PluxL({LuxR},{0.1},{bacterium}) from promoterLibrary.plb
    # Post-transcriptional processes associated with LuxR#
    PostTransc({LuxR},{3.2,0.3,0.04,3.6,0.075},{bacterium}) from postTranscriptionalLibrary.plb
    # Dimerisation of LuxR in the presence of 3OC6#
    DimSig({LuxR,3OC6,LuxR2},{1,0.0154,1,0.0154},{bacterium}) from postTranscriptionalLibrary.plb
  endRuleSet
endRulesets
endSPsystem

For technical reasons, our model using an extra cell to represent the boundary of the system:

SPsystem Boundary
  alphabet
    signal3OC6
  endAlphabet
  compartments
    cell
  endCompartments
The geometry of a bacterial colony of the cell type or bacterial strain represented in the previous model is captured using the following rectangular lattice:

```
# Author: Francisco J. Romero-Campero #
# Date: February 2010 #
# Description: A rectangular lattice of size 11x31 #

lattice rectangularLattice

# Dimension of the lattice and lower/upper bounds #
dimension 2
xmin 0
xmax 10
ymin 0
ymax 30

# Parameters used in the definition of the rest of components defining the lattice #
parameters
  parameter b1 value = 2
  parameter b2 value = 1
endParameters

# Basis vector determining the points in the lattice #
# in this case we have a rectangular lattice #
basis
  (b1,0)
  (0,b2)
endBasis

# Vertices used to determine the shape of the outmost membrane #
# of the SP systems located on each point of the lattice #
vertices
  (b1/2,b2/2)
  (-b1/2,b2/2)
  (-b1/2,-b2/2)
  (b1/2,-b2/2)
endVertices

# Vectors pointing at the neighbours of each point of the lattice #
neighbours
  (1,0)  (1,1)  (0,1)  (-1,1)
  (-1,0)  (-1,-1) (0,-1)  (1,-1)
endNeighbours
```
Finally, the model of the synthetical bacterial colony is obtained by distributing cellular clones of the sender cell strain at one end of the lattice and cellular clones of the pulsing cell strain over the rest of the points. This is modelled using the LPP-system below:

```
# Author: Francisco J. Romero-Campero #
# Date: February 2010 #
# Description: This LPP-system models the spatial distribution of the different #
# cellular strains in the synthetic bacterial colony developed by #
# Ron Weiss’ group exhibiting pulse propagation#
LPPsystem pulsePropagation

# List of the different bacterial strains used in the system #
SPsystems
    SPsystem senderCell from senderCell.sps
    SPsystem pulsingCell from pulsingCell.sps
    SPsystem boundaryCell from boundaryCell.sps
endSPsystems

# The geometry of the system is determined using a regular finite point lattice #
# spatialDistribution

# The boundary of the system is filled with boundary cells #
positions for boundaryCell
    parameters
        parameter i = 0:1:30
        parameter j = 0:10:10
    endParameters
    coordinates
        x = i
        y = j
    endCoordinates
endPositions

positions for boundaryCell
    parameters
        parameter i = 0:30:30
        parameter j = 1:1:9
    endParameters
    coordinates
        x = i
        y = j
    endCoordinates
endPositions

    # Sender cells are distributed at one end of the lattice #
    positions for senderCell
        parameters
            parameter i = 1:1:5
            parameter j = 1:1:9
        endParameters
        coordinates
            x=i
            y=j
```

endLattice
3.2.3 Simulations

Stochastic simulations of our model of the pulse generator can be run using the Infobiotics workbench. For this, please load using the provided interface the simulation parameter file, pulsePropagation.params, provided with the files comprising this example. Be patient, these simulations could take around five minutes.

Below we show a picture of the spatial propagation of a pulse of GFP over the bacterial colony. You can see our video here or here.
3.2.4 Model Checking

We observe that the further away the pulsing cells are from the sender cells the less likely they are of producing a pulse. We analyse this property by model checking the temporal formula below for a single pulsing cell recieving the signal at different rates:

\[ R = ? [ I = T ] T = 0:5:200 \]

3.3 Automatic Discovery of Pulse Generators

3.3.1 Introduction

The feed-forward loop (FFL) is a well studied network motif in systems biology. This model implements a pulse generator based on an incoherent type-1 FFL:

For this common FFL motif the transcription activator \( X \) activates gene \( Z \) directly, and also activates the repressor \( Y \) of the gene \( Z \), generating a pulse on the expression of \( Z \). The complete model can be downloaded from this link.

3.3.2 Library of Modules

The library of modules defines all modules that can be part of candidate models. This is essentially the list of buildings blocks from where the model will be constructed:

```plaintext
libraryOfModules FFL_library

# A module representing the unregulated expression of a gene X #
UnReg({X},{c_1 0:0.1:10 linear, c_2 0:0.01:2 linear, c_3 0:0.01:2 linear, c_4 0:0.01:2 linear}),
{
  rules:
  # Transcription of geneX #
  r1: [ geneX ]_l -c_1-> [ geneX + rnaX ]_l
  # Degradation of the RNA #
  r2: [ rnaX ]_l -c_2-> [ ]_l
  # Translation of the RNA #
```
r3: [ rnaX ]_l -c_3-> [ rnaX + proteinX ]_l
# Degradation of the protein #

r4: [ proteinX ]_l -c_4-> [ ]_l
}

# A module representing the positive regulation of a protein X #
# over a gene Y #
PosReg({X,Y},{c_1 0:0.1:10 linear,c_2 -3:1:3 logarithmic,c_3 0:0.1:10 linear, c_4 0:0.01:2 linear},{l}) =
{
  rules:
  # Binding and debinding of the transcription factor proteinX to geneY #
  r1: [ proteinX + geneY ]_l -c_1-> [ proteinX_geneY ]_l
  r2: [ proteinX_geneY ]_l -c_2-> [ proteinX + geneY ]_l
  # Transcription of geneY when proteinX is bound to its promoter #
  r3: [ proteinX_geneY ]_l -c_3-> [ proteinX_geneY + rnaY ]_l
  # Degradation of the RNA #
  r4: [ rnaY ]_l -c_4-> [ ]_l
  # Translation of the RNA #
  r5: [ rnaY ]_l -c_5-> [ rnaY + proteinY ]_l
  # Degradation of the protein #
  r6: [ proteinY ]_l -c_6-> [ ]_l
}

# A module representing the negative regulation of a protein X #
# over a gene Y #
NegReg({X,Y},{c_1 0:0.1:10 linear, c_2 -3:1:3 logarithmic},{l}) =
{
  rules:
  # Binding and debinding of the transcription factor proteinX to gene Y #
  r1: [ proteinX + geneY ]_l -c_1-> [ proteinX_geneY ]_l
  r2: [ proteinX_geneY ]_l -c_2-> [ proteinX + geneY ]_l
}

endLibraryOfModules

3.3. Automatic Discovery of Pulse Generators 97
3.3.3 Target Model

The target model is composed by the following modules:

\[ \text{UnReg}(X=\text{gene1}) \]
\[ \text{PosReg}(X=\text{gene1}, Y=\text{gene2}) \]
\[ \text{PosReg}(X=\text{gene1}, Y=\text{gene3}) \]
\[ \text{NegReg}(X=\text{gene2}, Y=\text{gene3}) \]

The model is optimised to output the target behaviour given by the time series for the \textit{rna} and \textit{protein} levels. The fitness is calculated based on the RMSE between the model output and the target data. As you can see below the expression of gene 3 generates a pulse.

3.3.4 Optimisation

The optimisation procedure can take some hours depending on the dimension of the search space. This include number of modules, possible instantiations for each module, and number of associated parameters (along with their ranges and precision). To speedup this process you can relax some of the optimisation parameters (population size, number of generations, number of simulation runs, proportion of models for parameter optimisation) at the cost of less accurate final models.

3.4 Auxin Transport

3.4.1 Introduction

This model addresses the transport of the hormone auxin through a file of plant cells. Auxin plays a major role in many aspects of plant growth and development. It moves through the plant in a polar manner due to non-uniform spatial distributions of active influx and efflux carriers on the cell membranes, and the resulting auxin distributions influence a wide range of processes, including organ initiation, vein formation and gravitropism. Modelling auxin transport is thus an active research area in plant systems biology. The models are inherently multiscale, as cell-scale processes lead to tissue-scale phenomena. To date, the majority of modelling in this area computes solutions by simulating large systems of deterministic ordinary differential equations, and there are relatively few examples of alternative modelling techniques. The model with a single file of cells can be seen below:
This stochastic computational model simulates the interaction of auxin at a molecular scale and, by analysing the gross movement of auxin from one compartment to the next, allow us to determine auxin dynamics at the tissue scale based on the mechanistic interactions of auxin at the molecular scale.

### 3.4.2 The model

The model is specified in SBML standard and can be downloaded here. For more information about the model, you should refer to:


A pdf copy of this paper is also available with the model file. This model is also available at EBI BioModels database (ID: MODEL1005200000) and JWS Model Repository.
If you intend to use results produced with the Infobiotics Workbench, please consider citing the following publications:

### 4.1 Related Publications


The Infobiotics Workbench is funded by the following grants:

(Semi)Formal Artificial Life Through P-Systems and Learning Classifier Systems: An Investigation into InfoBiotics (EP/E017215/1)

BB/D0196131 Centre for Plant Integrative Biology