

Visual GEC Manual

User manual version 0.10 beta

Michael Pedersen & Andrew Phillips

Introduction

Visual GEC is a tool for the design and simulation of transcriptional genetic circuits, or *devices*. The tool is based on GEC, a language for *Genetic Engineering of Cells*, which is described in detail in the following paper:

Michael Pedersen and Andrew Phillips: **Towards programming languages for genetic engineering of living cells**. In Journal of the Royal Society Interface, 15 April 2009.

GEC builds on previous research in the field of synthetic biology, including the notion of a Registry of Standard Parts (partsregistry.org, henceforth referred to as the Registry) together with experimental techniques for combining these parts into higher-level devices.

A range of related software tools have recently been developed. The main innovation behind GEC is to take the design process a step further, by allowing devices to be designed at a high level of abstraction with little or no knowledge of the specific parts available. The designer needs only a basic knowledge of the available part types, namely promoters, ribosome binding sites, protein coding regions and terminators. Parts of these elementary types can be composed to form genes and gene networks. The higher level of abstraction in GEC is achieved by specifying constraints between otherwise unspecified parts: the GEC compiler automatically determines a solution with the actual parts that satisfy the design constraints.

In most cases, multiple solutions are possible for a given design. GEC can compile each of the solutions to a set of chemical reactions, which can then be simulated or analyzed by the designer. The solutions that exhibit the desired behaviour can then be synthesized and put to work in living cells. Although there is no guarantee that a solution which produces the desired simulation results will function correctly inside a living cell, analyzing the design on a computer is an effective way to rapidly detect design errors prior to building the physical device - a process which can take several days and for which even small errors can prove very costly.

In this manual we first describe how to access Visual GEC on the web. We then walk through the user interface, from the parts database to the GEC program editor and simulator. This should allow new designers to get up to speed using Visual GEC based on the built-in examples. We then proceed with an informal overview of the GEC language itself based on small examples. We end in the last section with a presentation of the concrete syntax of GEC. For further details, including a formal presentation of the semantics of GEC, please refer to the published paper.

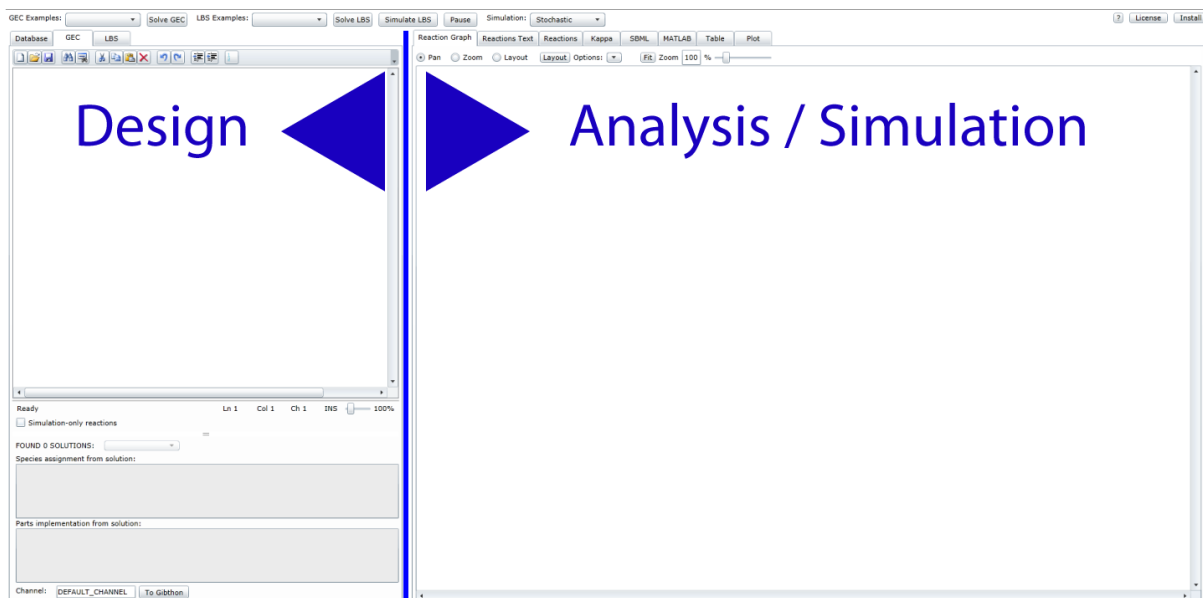
Accessing Visual GEC

Visual GEC is a Silverlight application and requires the Silverlight plug-in for your operating system and web browser to be installed from <http://www.microsoft.com/silverlight>. Silverlight compatibility has been tested on Windows and Mac OS X under various browsers, including Internet Explorer, Firefox, Safari and Chrome.

With Silverlight installed, browse to <http://lepton.research.microsoft.com/webgec> and the user interface should load straight away. Visual GEC can optionally be installed locally by pressing the “Install” button in the top-right corner of the user interface. This will allow direct access to Visual GEC from your computer without any need for Internet access. To the right of the “Install” button is a “License” button which brings up a copy of the Visual DSD license agreement. Once the software is installed, the “Install” button becomes an “Update” button which can be used to check for, and install, any newer releases of the software.

Note that when a new version of the software is released online, you may need to “reset” your web browser to delete the old version from the cache before your browser will load the new version.

The main screen of the Visual GEC contains two tab panels. The panel on the left is concerned with design, and the panel on the right is concerned with analysis and simulation. We continue in the next section by describing each of the left hand side design tabs in turn.



User interface walk-through: the design tabs

The Database tab

Visual GEC interface showing the Database tab. The interface includes dropdown menus for 'GEC Examples' and 'LBS Examples', and buttons for 'Solve GEC' and 'Solve LBS'. Below these are tabs for 'Database', 'GEC', and 'LBS'. The 'Database' tab is active, showing 'Reset', 'Save', and 'Load' buttons. It contains two tables: 'Parts database' and 'Reactions database'.

Parts database:

Enabled	ID	Type	Properties	Comments
<input checked="" type="checkbox"/>	i723025	pcr	codes(phzS, 0.001)	
<input checked="" type="checkbox"/>	c0040	pcr	codes(tetR, 0.001)	
<input checked="" type="checkbox"/>	c0080	pcr	codes(araC, 0.001)	
<input checked="" type="checkbox"/>	cunknown4	pcr	codes(ccdA, 0.1)	
<input checked="" type="checkbox"/>	i723020	prom	pos(toluene-xylR, 0.001, 0.001, 1.0), con(0.0001)	
<input checked="" type="checkbox"/>	r0051	prom	neg(clR, 1.0, 0.5, 0.00005), con(0.12)	
<input checked="" type="checkbox"/>	r0040	prom	neg(tetR, 1.0, 0.5, 0.00005), con(0.09)	
<input checked="" type="checkbox"/>	runknown2	prom	pos(lasR-m3OC12HSL, 1.0, 0.8, 0.1), pos(luxR-m3OC6HSL, 1.0, 0.8, 0.1)	
<input checked="" type="checkbox"/>	b0034	rbs	rate(0.1)	
<input checked="" type="checkbox"/>	b0015	ter		

Reactions database:

Enabled	Reactions	Comments
<input checked="" type="checkbox"/>	toluene + xylR ->{1.0} toluene-xylR	
<input checked="" type="checkbox"/>	phzM ~ pca ->{1.0} metPCA	
<input checked="" type="checkbox"/>	phzS ~ metPCA ->{1.0} pyo	
<input checked="" type="checkbox"/>	luxR + m3OC6HSL ->{0.5} luxR-m3OC6HSL	

Visual GEC comes bundled with sample databases which can be used as a basis for experimenting with the bundled sample models. The databases can be modified, and any changes can be saved to and loaded from a file by pressing the "Save" and "Load" buttons, respectively. Note that changes to the databases are not automatically saved, so saving must be done manually in order to avoid losing changes. The original bundled sample databases can be restored by pressing the "Reset" button.

There are two databases, one for parts (at the top) and one for reactions (at the bottom). In both cases, a new row can be added by pressing the "add" button, and an existing, selected row can be deleted by pressing the "Delete" button.

The parts database has the following columns:

- **Enabled.** When checked, this indicates that a part is enabled. If it is not, it will be ignored by the tool. This can be useful for experimenting with and debugging a model.

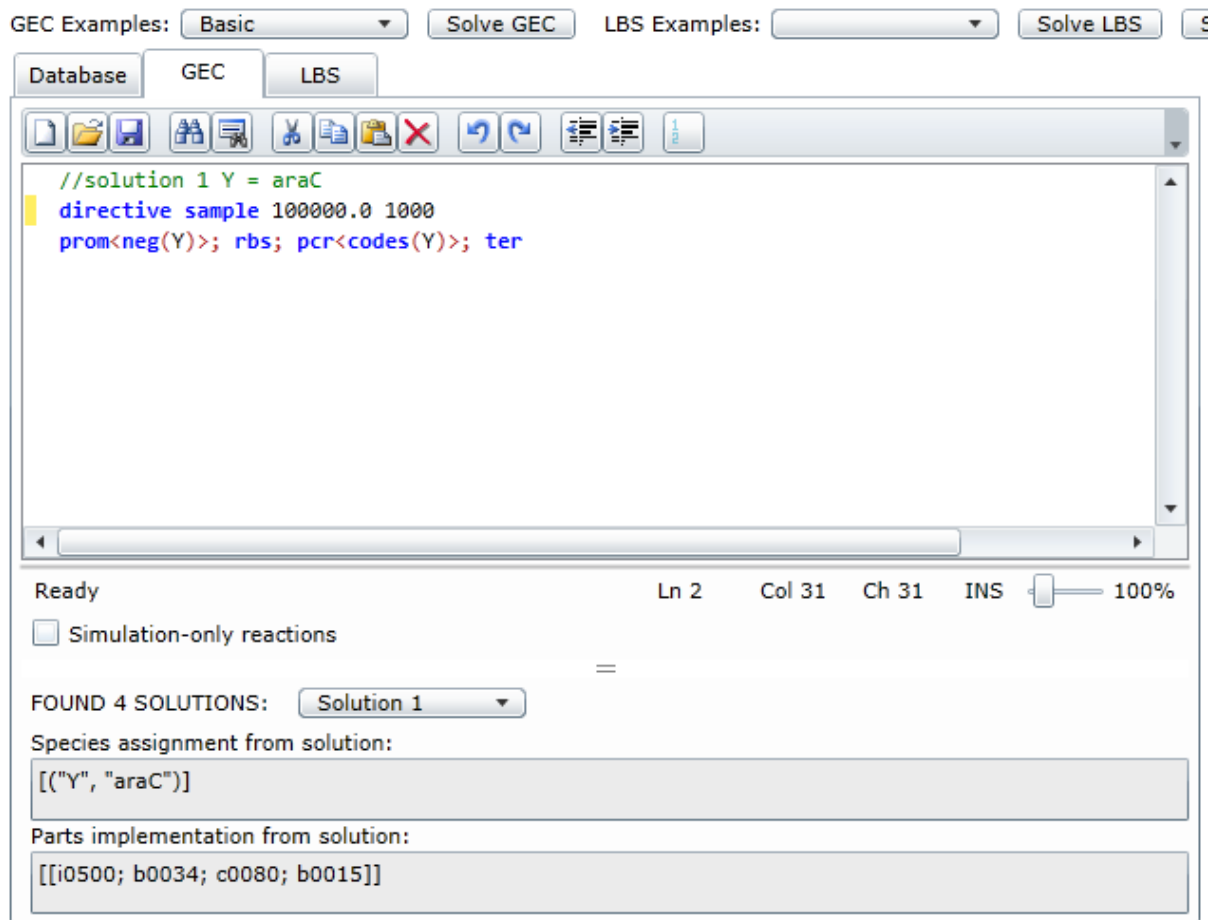
- **ID.** This identifies the part in the database and must be unique. In the sample database, most of the IDs refer to those of the Registry.
- **Type.** This indicates whether a part is a promoter ("prom"), a ribosome binding site ("rbs"), a protein coding region ("pcr") or a terminator ("ter").
- **Properties.** Properties constitute the characterisation of a part. They are used both for resolving constraints in a GEC model, and for constructing the reactions to be used for simulation.
 - **Terminators** currently have no properties.
 - **Ribosome binding sites** have a property of the form "rate(0.1)" which represents a rate of translation of mRNA arising from upstream parts to proteins arising from downstream parts.
 - **Protein coding regions** have a property of the form "codes(tetR, 0.1)" where the first component is the protein coded by the part (here tetR), and the second component is the degradation rate of this protein.
 - **Promoters** typically have several properties. The *constitutive* property "con(0.0001)" represents the constitutive rate of transcription from the promoter. The *negative* property of the form "neg(tetR, 1.0, 0.5, 0.00005)" states that the promoter is negatively regulated by the given transcription factor, here tetR. The remaining real-numbered components represent the rate of promoter-transcription factor binding, the rate of unbinding, and the rate of transcription in the bound state. There is a corresponding *positive* property of the form "pos(toluene-xyIR, 0.001, 0.001, 1.0); in this case, the transcription factor is a complex. From these properties, the tool automatically deduces a number of reactions which explicitly model transcription at the level of transcription factor binding.
- **Comments.** This can be used to associate any further information with a part.

The reactions database is included as a proof of concept, and is intended to represent a comprehensive knowledge base of possible reactions. These can be used to further constrain parts in a model, e.g. by requiring that the proteins expressed by two protein coding regions can dimerise, or be transported in and out of a cell. In practice, however, the reactions database can be ignored for most purposes: reactions which are needed for simulation can be included directly in a model in the GEC editor.

The reactions database has "Enabled" and "Comments" columns similar to the parts database. It also has a "Reactions" column which describes the actual reactions, of the form: "enzyme ~ s₁ + ... + s_i -> p₁ + ... + p_j". The enzymatic part can optionally be omitted. Transport reactions for representing transport in and out of a cell are of the form "s -> c[s]" and "c[s] -> s", respectively, where "s" is a species and "c" denotes some compartment; the compartment name is insignificant in the context of the database.

Be aware that all numbers in the databases must be written with a decimal point (e.g. "1.0" instead of "1"), or the database will silently fail to work.

The GEC tab



The GEC tab provides functionality for editing and compiling GEC programs. The top section of the tab contains a code editor. The editor has a row of buttons with standard editing functionality, e.g. for saving and opening GEC files. Hover the mouse over a button for a description of its functionality. Directly below the editor are numbers showing the line and column numbers of the cursor, and a slider for changing the font size as needed. The "Simulation-only reactions" box indicates that any reactions in a model should be used for simulation only, and should not be considered as constraints of the model. We discuss this issue further in the context of the GEC language.

Visual GEC comes bundled with a number of sample GEC programs which can be selected from the drop-down box labeled "GEC Examples". The "Basic" example shown in the screenshot, for instance, is a simple model of a negative feedback loop. The first line of this example is a "//"-prefixed single-line comment. Multi-line comments are also possible: these are opened with "/*" and closed with "*/".

The second line of the Basic example, starting with "directive", has no bearing on the model itself, but rather specifies simulation parameters. The following summarizes the available directives:

- **Sample** directives are of the form "directive sample 10000.0 1000", as in the screenshot. This specifies that a simulation should run for 10000.0 time units (the first number) and that 1000 data points should be collected for plotting (the second number). Increasing the number of data points in the same period of simulation time produces more fine-grained

results but the display may be less responsive. Similarly, if the number of data points stays constant but the simulation time is extended (or shortened), the resulting plot will be less (or more) detailed. If no sample directive is provided, the default behaviour is to run the simulation for 1000 time units and take 1000 samples of the species populations. Be aware of a current restriction: the simulation time must be specified as a floating point number, or the compiler will complain.

- **Plot** directives are of the form "directive plot A; B; C". This specifies which species to plot during simulation; see the bundled Repressilator model for an example. Take care to spell the species names correctly. If for example compartments are included in the model, the correct compartment should also be used in the plot directive, and must be applied per-species; for example, write "c[A]-c[B]" rather than "c[A-B]". Also be aware of a current bug: if a complex species is plotted (e.g. by a statement of the form "directive plot A-B"), the order of species A and B sometimes matters. So if no simulation output appears, changing the order of species in a complex might resolve the problem.
- **Scale** directives allow the stochastic simulator to scale up from molar concentrations to populations of individuals. Concentrations are scaled by simply multiplying by the factor and the rates of binary reactions are modified following Section 4.2 of (Cardelli, 2008). Thus the user does not have to worry about the details involved in switching between continuous and discrete simulation methods (see below). The scale factor is 1.0 by default. The scale factor also modifies the tolerance parameter of the deterministic simulator, as described below. See the "Populations and concentrations" section below for further details.
- **Time** directives allow one to specify the assumed unit of time. It will be printed on the x-axis of simulation plots. The default time units are seconds (s).
- **Concentration** directives allow one to specify the assumed unit of concentrations. It will be printed on the y-axis of simulation plots when the simulator is run in deterministic mode. The default concentration units are nanomolar (nM).
- **Tolerance** directives are of the form "directive tolerance 0.001". This specifies the tolerance parameter of the deterministic simulator, which provides a tradeoff between computational cost and smoothness of the resulting solution. The default value is 10^{-6} . It is crucial to choose a tolerance value which reflects the populations and reaction rates of the system in question, or the performance of the deterministic simulator may suffer. Note that the tolerance is multiplied by the scale factor in an attempt to maintain a reasonable value with respect to the species populations.

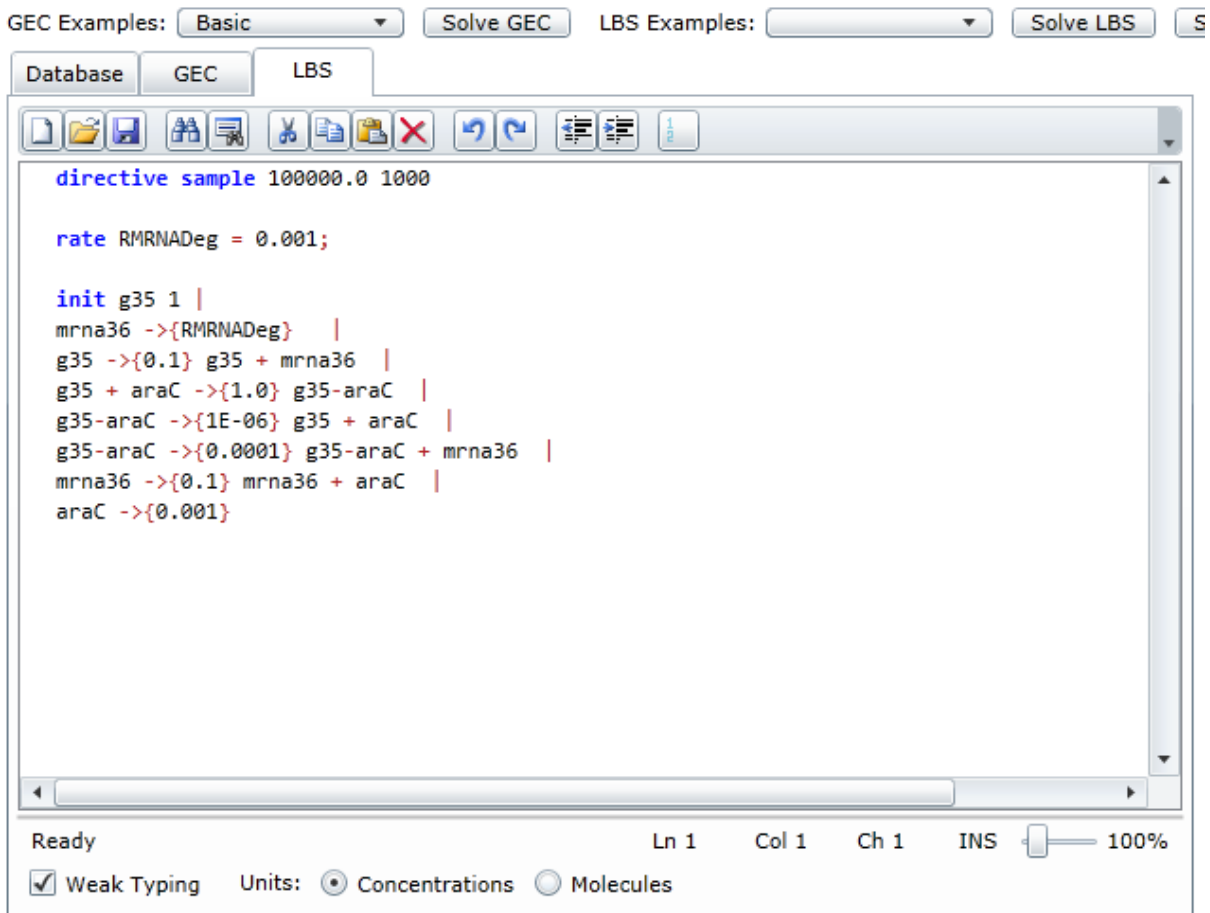
Line 3 of the Basic example in the above screenshot is the core of the model, representing a single gene expressing a protein, Y, which negatively regulates the promoter. We explain the GEC language itself in more detail later. For now, the point to note is that the model contains unspecified parts with certain properties of the kind described in the database section above. A property may contain variables (here Y), and there may be implicit constraints on these variables (here that the same Y is expressed by the protein coding region *and* represses the promoter).

A model can be solved by pressing the "Solve GEC" button. If there are any errors, a message box appears with an indication of the cause. Otherwise, the number of possible solutions is displayed below the editor, with a drop-down box allowing individual solutions to be selected. Note that there

may be no solutions if there are no appropriate parts in the database satisfying the constraints of the model.

Selecting a solution populates the "Species assignment" and "Parts implementation" boxes. The species assignment box shows which species have been assigned to variables. For instance, [{"Y", "araC"}] indicates that araC has been assigned to the variable Y for a given solution in the Basic example. The parts implementation box shows which concrete parts have been chosen for a given solution. For instance, [{"i0500; b0034; c0080; b0015"}] indicates that i0500 is the chosen promoter; that b0034 is the chosen ribosome binding site; that c0080 is the chosen protein coding region; and that b0015 is the chosen terminator in the case of the Basic example. The parts are selected from the database, and the part names are their database IDs.

The LBS tab



The screenshot shows the LBS editor interface. At the top, there are dropdown menus for "GEC Examples: Basic" and "LBS Examples:", along with "Solve GEC" and "Solve LBS" buttons. Below these are tabs for "Database", "GEC", and "LBS". The "LBS" tab is active, displaying a text editor with the following LBS model code:

```
directive sample 100000.0 1000

rate RMRNADeg = 0.001;

init g35 1 |
mrna36 ->{RMRNADeg} |
g35 ->{0.1} g35 + mrna36 |
g35 + araC ->{1.0} g35-araC |
g35-araC ->{1E-06} g35 + araC |
g35-araC ->{0.0001} g35-araC + mrna36 |
mrna36 ->{0.1} mrna36 + araC |
araC ->{0.001}
```

The editor status bar at the bottom shows "Ready", "Ln 1", "Col 1", "Ch 1", "INS", and a zoom level of "100%". There are also checkboxes for "Weak Typing" and radio buttons for "Units: Concentrations" (selected) and "Molecules".

When a solution is chosen in the GEC tab, a corresponding reaction model is generated for simulation. This model is written in a language called LBS (a *Language for Biochemical Systems*) and the model appears automatically under the LBS tab. In many cases the LBS model can be used as-is, without any modification, but it is made available for cases where further customization or model reduction is needed.

The LBS editor itself is similar to that for GEC, and so are the directives (e.g. "plot" and "sample") which are copied verbatim from the GEC model. The LBS model generally begins with a rate

definition which is used for all generated mRNA degradation reactions (“rate RMNADeg = 0.001”). The reason is that mRNA in the general case can be polycistronic, so mRNA degradation rates cannot easily be included directly in the parts database. If different mRNA degradation rates are needed for individual reactions, these can be entered manually in the LBS model. The global default rate can also be specified directly in the GEC model, where it takes the slightly different form “rateDef RMRNADeg 0.001”.

The LBS model generally contains a number of reactions separated by the “|” symbol. Rates are generally assumed to be mass-action and are enclosed by curly brackets (“{”, “}”) after a reaction arrow. If the GEC model contains compartments, so will the corresponding LBS model.

Below the LBS editor is a checkbox for "Weak typing". This instructs the LBS compiler to allow species to be used without first being declared. As the LBS output from a GEC solution does not declare species, the "Weak typing" box is checked by default when LBS is used in conjunction with GEC.

Below the LBS editor is also an option to select whether the species units are concentrations or molecules. This currently does not have any effect in the tool, but in future might be reflected in the encoding of the SBML output for a model.

Press the "Solve LBS" button for any changes in the LBS model to take effect in the right hand side tabs. Note that any unsaved changes made to the LBS model will disappear when selecting another solution under the GEC tab.

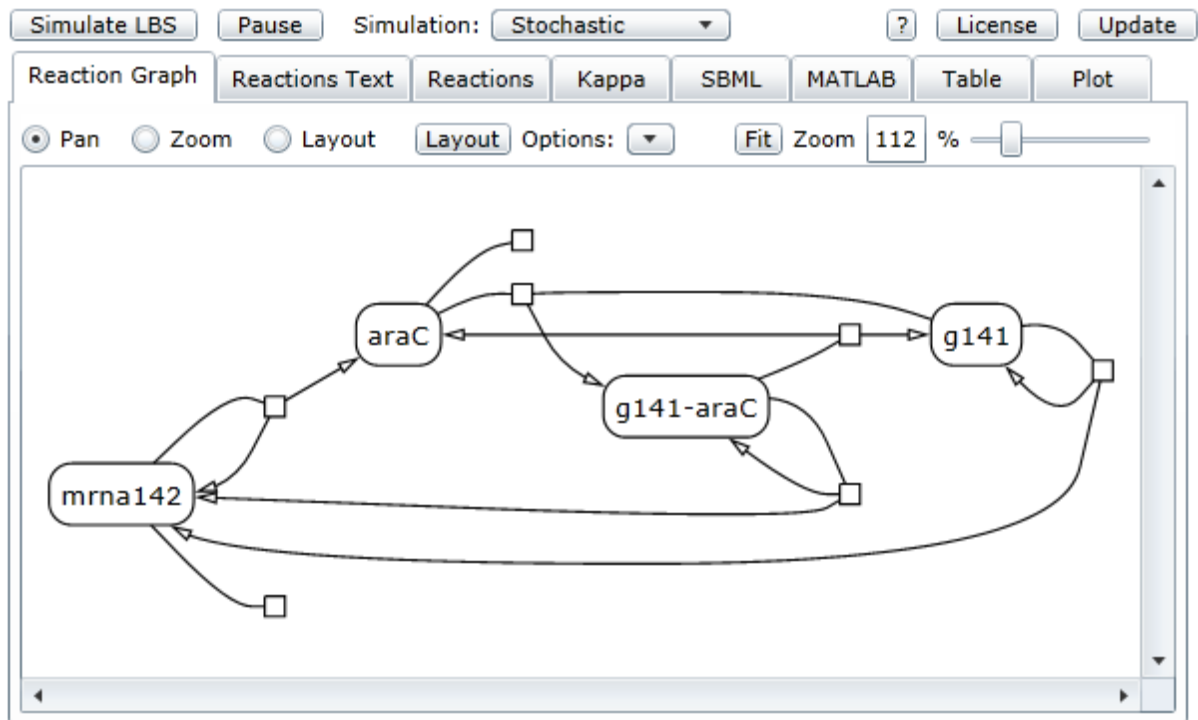
Some instructive LBS examples can be found by selecting from the "LBS Examples" drop-down box at the top of the screen. For a detailed presentation of the LBS language itself, please refer to e.g. the following published paper:

Michael Pedersen and Gordon D Plotkin: **A Language for Biochemical Systems: Design and Formal Specification**. In Proceedings of T. Comp. Sys. Biology. 2010, 77-145.

User interface walk-through: the analysis tabs

The right hand side of Visual GEC contains a number of tabs related to the analysis and simulation of a GEC solution. We describe each in turn.

The Reaction Graph tab



This tab shows a visual representation of the reactions for a solution. The square boxes in the picture represent reactions, and the rounded rectangles represent species. Lines with no arrowheads indicate the reactants of a reaction, and lines with arrowheads indicate the products. This visual representation is similar to that of the standard Petri net associated with a set of chemical reactions.

Towards the top of the tab are a number of visualisation options:

- The **Pan** button puts the mouse cursor in panning mode, allowing the graph to be moved around the screen.
- The **Zoom** radio button puts the mouse cursor in zoom mode, allowing a rectangular selection of the graph to be enlarged. In addition, the zoom slider to the right allows precise control of the zoom factor, and there is an option to fit the graph to the available screen space.
- The **Layout** radio button puts the mouse cursor in layout modification mode, allowing individual reactions and species to be moved by clicking and dragging them to the desired location. To revert to the automatic layout, press the Layout button at any time.
- The **Options** drop-down box allows the layout to be predominantly **horizontal** (the default is vertical); the **aspect ratio** to be confined to that of the graph window; and the **rates shown** in reaction boxes (rates are hidden by default).

The Reaction Text tab

The screenshot shows the 'Reaction Text' tab selected. At the top, there are buttons for 'Simulate LBS', 'Pause', and 'Simulation: Stochastic'. Below these are tabs for 'Reaction Graph', 'Reactions Text', 'Reactions', 'Kappa', 'SBML', 'MATLAB', 'Table', and 'Plot'. A 'Zoom' slider and a 'Save' button are located above the main text area. The text area contains the following reactions:

```
mrna142 -->{0.001}
g141 -->{0.1} g141 + mrna142
g141 + araC -->{1} g141-araC
g141-araC -->{1E-06} g141 + araC
g141-araC -->{0.0001} g141-araC + mrna142
mrna142 -->{0.1} mrna142 + araC
araC -->{0.001}
```

This tab shows a textual representation of the reactions for a solution. Towards the top of the tab is a slider for adjusting the zoom level, and a button for saving the text to file.

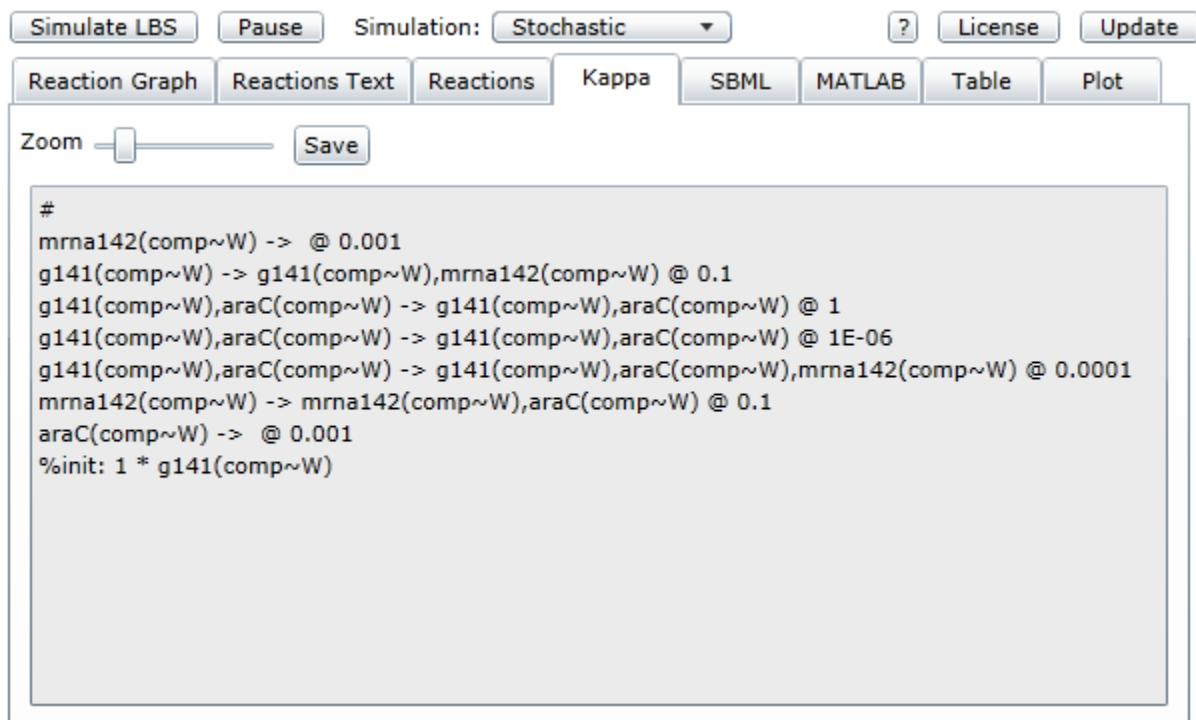
The Reactions tab

The screenshot shows the 'Reactions' tab selected. The interface is similar to the previous one, but the 'Reactions' tab is active. The reactions are displayed in a stylized format with graphical arrows and rate constants:

```
mrna142  $\xrightarrow{0.001}$ 
g141  $\xrightarrow{0.1}$  g141+mrna142
g141+araC  $\xrightarrow{1}$  g141-araC
g141-araC  $\xrightarrow{1E-06}$  g141+araC
g141-araC  $\xrightarrow{0.0001}$  g141-araC+mrna142
mrna142  $\xrightarrow{0.1}$  mrna142+araC
```

This tab shows a stylized representation of the reactions for a solution, in which reaction arrows are drawn graphically rather than textually as in the Reaction Text tab.

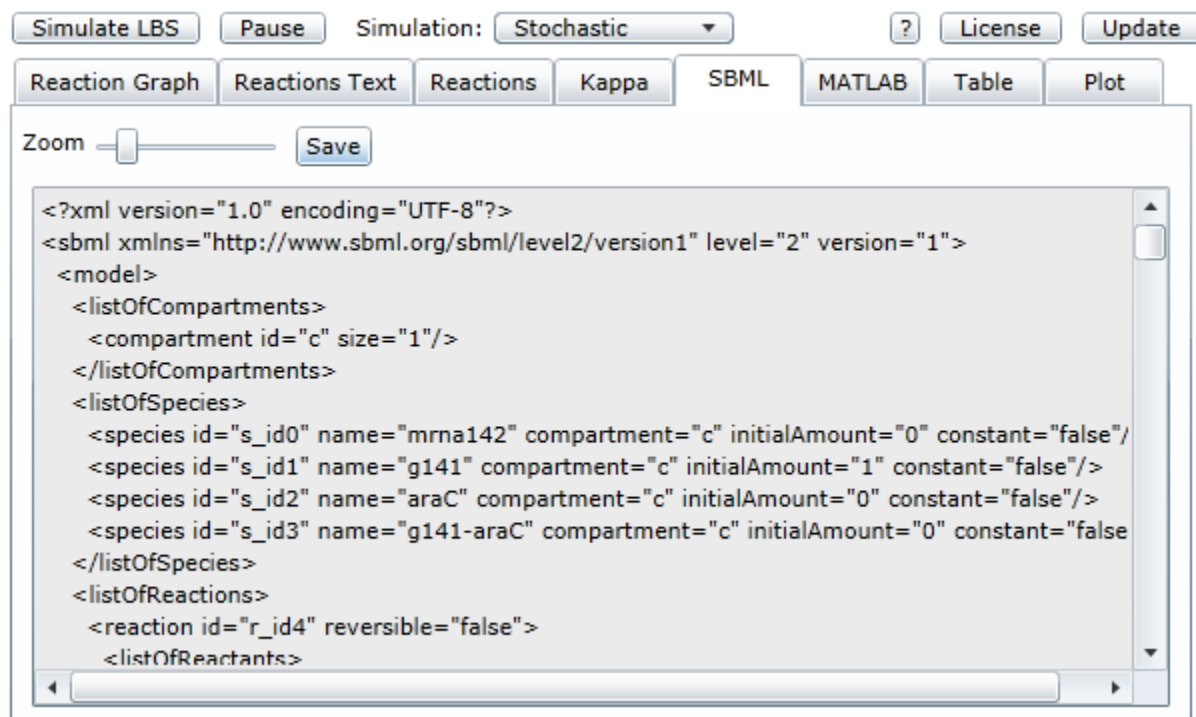
The Kappa tab



```
#
mrna142(comp~W) -> @ 0.001
g141(comp~W) -> g141(comp~W),mrna142(comp~W) @ 0.1
g141(comp~W),araC(comp~W) -> g141(comp~W),araC(comp~W) @ 1
g141(comp~W),araC(comp~W) -> g141(comp~W),araC(comp~W) @ 1E-06
g141(comp~W),araC(comp~W) -> g141(comp~W),araC(comp~W),mrna142(comp~W) @ 0.0001
mrna142(comp~W) -> mrna142(comp~W),araC(comp~W) @ 0.1
araC(comp~W) -> @ 0.001
%init: 1 * g141(comp~W)
```

This tab shows a representation of the reactions for a solution in the rule-based language Kappa. Note however that the kappa rules resulting from a GEC solution are not per se meaningful. To generate meaningful Kappa, special syntax must be used in the LBS model; hence the Kappa output is not directly relevant to modelling with GEC.

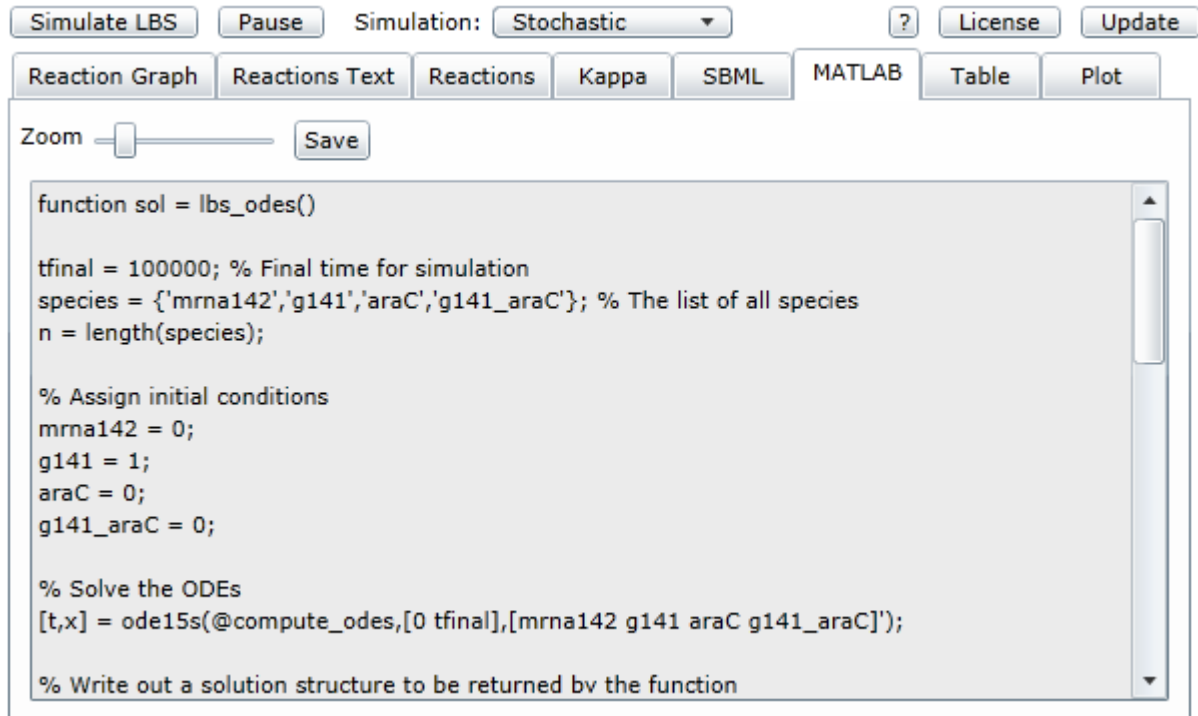
The SBML tab



```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2/version1" level="2" version="1">
  <model>
    <listOfCompartments>
      <compartment id="c" size="1"/>
    </listOfCompartments>
    <listOfSpecies>
      <species id="s_id0" name="mrna142" compartment="c" initialAmount="0" constant="false"/>
      <species id="s_id1" name="g141" compartment="c" initialAmount="1" constant="false"/>
      <species id="s_id2" name="araC" compartment="c" initialAmount="0" constant="false"/>
      <species id="s_id3" name="g141-araC" compartment="c" initialAmount="0" constant="false"/>
    </listOfSpecies>
    <listOfReactions>
      <reaction id="r_id4" reversible="false">
        <listOfReactants>
```

This tab shows an SBML (Systems Biology Markup Language) representation of the reactions for a solution. This can be used for exchange with other modelling and simulation tools. Towards the top of the tab is a slider for adjusting the zoom level, and a button for saving the text to file.

The MATLAB tab



The screenshot shows a software interface with a top navigation bar containing buttons for 'Simulate LBS', 'Pause', 'Simulation: Stochastic', '?', 'License', and 'Update'. Below this is a row of tabs: 'Reaction Graph', 'Reactions Text', 'Reactions', 'Kappa', 'SBML', 'MATLAB', 'Table', and 'Plot'. The 'MATLAB' tab is active, displaying a code editor with the following MATLAB code:

```
function sol = lbs_odes()

tfinal = 100000; % Final time for simulation
species = {'mrna142','g141','araC','g141_araC'}; % The list of all species
n = length(species);

% Assign initial conditions
mrna142 = 0;
g141 = 1;
araC = 0;
g141_araC = 0;

% Solve the ODEs
[t,x] = ode15s(@compute_odes,[0 tfinal],[mrna142 g141 araC g141_araC]');

% Write out a solution structure to be returned by the function
```

This tab shows a MATLAB representation of the reactions for a solution, which can be used for further simulation in MATLAB. Towards the top of the tab is a slider for adjusting the zoom level, and a button for saving the text to file.

Simulation

The reactions representing a given solution can be simulated by pressing the "Simulate LBS" button at the top of the screen, and the simulation can be stopped by pressing the "Pause" button. There are two possible simulation modes, namely stochastic and deterministic, which can be chosen by selecting from the "Simulation" drop-down box. The duration of the simulation, and the species to report, are specified using "directive" statements in the GEC and LBS code as previously described. Once the simulation is started, the result can be viewed in table form and in plot form as described below.

The Table tab

Simulate LBS Pause Simulation: Stochastic ? License Update

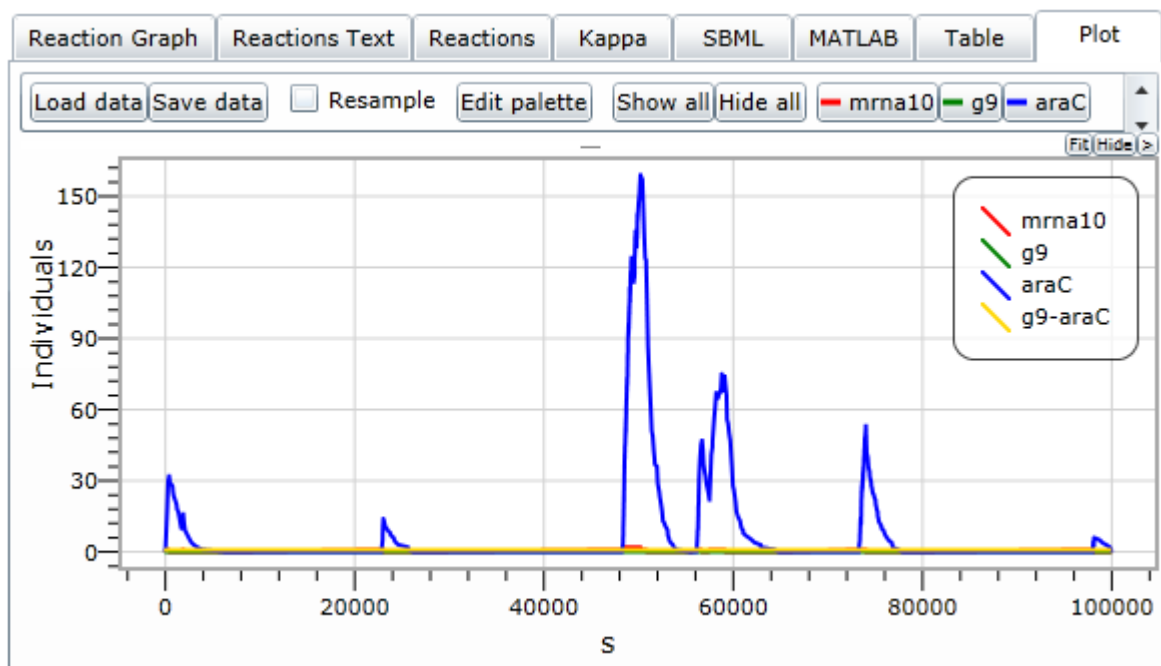
Reaction Graph Reactions Text Reactions Kappa SBML MATLAB Table Plot

Save Data Current page: 1 of 10 Items per page: 100 Total items: 978 Prev Page Ne

Time	mrna142	g141	araC	g141-araC
0	0	1	0	0
93.0210467415344	1	0	10	1
198.060764035913	1	0	16	1
290.759415029611	1	0	19	1
362.539064421561	1	0	23	1
495.174739832082	1	0	26	1
597.809294263634	1	0	35	1
680.669960928062	1	0	41	1
775.936735768081	1	0	48	1
860.090990888275	0	0	47	1

This tab shows the result of a simulation in table form. The table is generally restricted to a certain number of data points per page. Towards the top of the tab are options for specifying the page number and the number of data points per page; for choosing to show all data points in a single page; buttons for navigating between pages; and a button for saving the table to e.g. a Comma Separated File for visualisation or analysis by third party apps.

The Plot tab



This tab shows the result of simulation as a plot with time along the x-axis and population values along the y-axis. Towards the top of the tab are buttons for loading and saving plots to file. Checking the "Resample" check box reduces the number of data points plotted, which can improve performance. Each species is represented by a colour-coded button which can be pressed to toggle the plot for the corresponding species in the plot; all species can be shown by pressing the "Show all" button, and all species can be hidden by pressing the "Hide all" button.

Populations and concentrations

In Visual GEC, quantities are specified as molar concentrations, which denote the number of moles per unit volume. The units of concentration can be set by the `concentration` directive. For example, directive `concentration "M"` sets the units of concentration to molar. The default units are nanomolar (nM), where $1\text{nM} = 10^{-9} \text{ mol/L}$.

In order to perform a stochastic simulation, concentrations must be converted to numbers of individuals. This can be achieved using the following equation:

$$n = \lceil c \cdot V \cdot N_A \rceil$$

where n is the number of individuals, c is the molar concentration, V is the volume and N_A is Avogadro's constant, which denotes the number of individuals per mole of substance (approximately $6.02214 \times 10^{23} \text{ mol}^{-1}$). The function $\lceil x \rceil$ denotes the rounding up of x to its nearest natural number. Thus, in order to convert a concentration into a number of individuals, it is sufficient to multiply the concentration by a *scale factor* $s = V \cdot N_A$, which denotes the number of individuals per unit concentration. Essentially, this corresponds to choosing a volume V such that the number of individuals is equal to s for one unit of concentration. For example, a scale factor of 50 corresponds to choosing a volume that is 50 times the volume occupied by a single individual. The units of the scale factor are assumed to be the inverse of the units of concentration, and are given as nM^{-1} by default. Note that the conversion from concentrations to individuals is achieved using a scale factor s rather than specifying a volume V directly, since it is difficult to choose a volume such that the number of individuals is a natural number. The scale factor can be set by the `scale` directive, where the default scale factor is 1.0.

The choice of deterministic (continuous) or stochastic (discrete) simulation is also manifested in the units for the simulation plot. The vertical axis of the plot has units of individuals for stochastic simulation, and units of concentration for deterministic simulation. Note that the units for rate constants are assumed to be consistent with the units for time and concentration. For example, if the units for time are s and the units for concentration are nM, then the units for the bimolecular rate constants are assumed to be $\text{nM}^{-1}\text{s}^{-1}$, and the units for the unimolecular rate constants are assumed to be s^{-1} . Once a suitable scale factor has been selected, in order to perform a stochastic simulation the molar concentrations are multiplied by the scale factor, while the concentration-dependent rates are divided by the scale factor. For example, if the scale factor is 100 nM^{-1} then a concentration-dependent rate of $0.4 \text{ nM}^{-1}\text{s}^{-1}$ is converted to a stochastic rate of 0.004 s^{-1} for simulation. Additional details on converting between populations and concentrations can be found in Section 4.2 of (Cardelli, 2008), including specific conversion rules for homodimerization reactions.

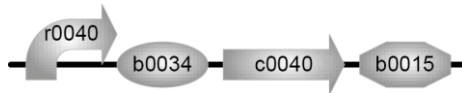
An informal overview of the GEC language

This section gives a brief and informal introduction to the GEC language itself through examples.

Part types

On the most basic level, a model can simply be a sequence of part IDs together with their types. The following model is an example of a transcription unit which expresses the protein tetR in a negative feedback loop; the corresponding graphical representation shown beneath.

```
r0040:prom ; b0034:rbs ; c0040:pcr ; b0015:ter
```



The symbol ":" is used to write the type of a part, and the symbol ";" is the sequential composition operator used to put parts together in sequence (conceptually on the same piece of DNA). Writing this simple model requires the modeller to know that the protein coding region part c0040 codes for the protein tetR and that the promoter part r0040 is negatively regulated by this same protein, two facts which we can confirm by inspecting the default parts database bundled with Visual GEC. In this case the compiler has an easy job: it just produces a single list consisting of the given sequence of part IDs, while ignoring the part types:

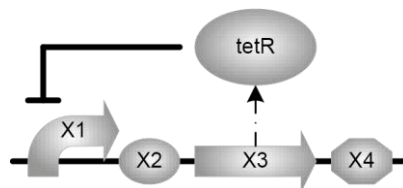
Parts implementation from solution:

```
[[r0040; b0034; c0040; b0015]]
```

Part variables and properties

We can increase the abstraction level of the model by using *variables* and *properties* for expressing that any parts will do, as long as the protein coding region codes for the protein tetR and the promoter is negatively regulated by tetR:

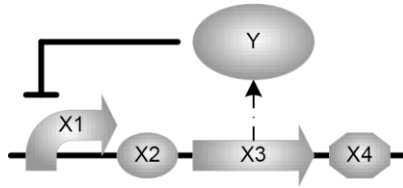
```
X1:prom<neg(tetR)> ; X2:rbs ; X3:pcr<codes(tetR)> ; X4:ter
```



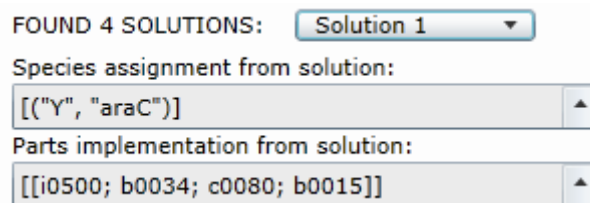
As in the database, the angle brackets <> delimit one or more properties, and upper-case names such as X1 represent variables (undetermined part names or species). Compiling this model produces exactly the same result as before, only this time the compiler does the work of finding the specific parts required based on the information stored in the parts database.

The compiler may in general produce several results. For example, we can replace the fixed species name tetR with a new variable, thus resulting in a program expressing *any* transcription unit behaving as a negative feedback device:

```
X1:prom<neg(Y)> ; X2:rbs ; X3:pcr<codes(Y)> ; X4:ter
```

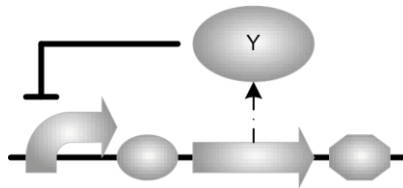


This time the compiler produces 4 solutions given the standard database, one of them being the tetR device from above. Choosing one solution now also populates the "Species assignment" section:



When variables are only used once, as is the case for X1, X2, X3 and X4 above, their names are of no significance and we will use the *wild card*, `_`, instead. When there is no risk of ambiguity, we may omit the wild card altogether and write the above program more concisely as follows:

```
prom<neg(Y)> ; rbs ; pcr<codes(Y)> ; ter
```

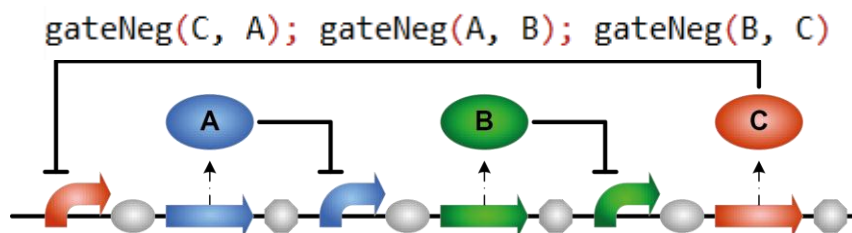


Parameterised modules

Parameterised modules can be used to add a further level of abstraction to a model. For example, a module which acts as a negative gates can be written as follows, where "i" denotes *input* and "o" denotes *output*:

```
module gateNeg(i,o) {
  prom<neg(i)>; rbs; pcr<codes(o)>; ter
};
```

Using this module, the *repressilator* circuit, in which three genes repress each other, can be written concisely as follows:



In general, the "module" keyword is followed by the name of the module, a list of formal parameters, and the body of the module enclosed in brackets; a module can be invoked simply by naming it followed by a list of actual parameters in parentheses.

The repressilator model yields 24 solutions based on the sample database:

FOUND 24 SOLUTIONS:

Solution 1 ▾

Species assignment from solution:

```
[("A", "lacI"); ("B", "tetR"); ("C", "araC")]
```

Parts implementation from solution:

```
[[i0500; b0034; c0012; b0015; r0011; b0034; c0040; b0015; r0040; b0034; c0080; b0015]]
```

Compartments and reactions

Compartments can be used to represent the location of devices in the case where a multi-cellular system is being designed. For example, the following contrived model can be thought of as a multi-cellular repressilator, with each gene located in different cells c1, c2 and c3:

```
c3[gateNeg(C,A)] || c1[gateNeg(A,B)] || c2[gateNeg(B,C)]
```

Because the genes are in different cells and do not follow each other on the same piece of DNA, the *parallel composition* operator "||" is used rather than the sequential composition operator ";" previously used. As a result, the parts implementation no longer consists of a single list of parts, but instead of three lists, reflecting the fact that the genes are physically separated on different strands of DNA:

FOUND 24 SOLUTIONS:

Solution 1 ▾

Species assignment from solution:

```
[("A", "lacI"); ("B", "tetR"); ("C", "araC")]
```

Parts implementation from solution:

```
[[i0500; b0034; c0012; b0015]; [r0011; b0034; c0040; b0015]; [r0040; b0034; c0080; b0015]]
```

Reactions, which may include compartments, can be used to impose additional constraints on parts. For example, we might impose the additional constraints that the expressed proteins can move between cells, and, arbitrarily, that proteins B and C can dimerise under the catalysis of A:

```

c3[gateNeg(C,A)] || c1[gateNeg(A,B)] || c2[gateNeg(B,C)]
| A -> c2[A] | c1[A] -> A
| B -> c3[B] | c2[B] -> B
| C -> c1[C] | c3[C] -> C
| A ~ B + C -> B-C

```

The constraints are composed with models using the *constraint composition* operator, "|". For this particular example, there are no solutions based on the sample database. If the reactions are not intended as constraints, they can be used purely for simulation by appending a star to the reaction arrows as follows:

```

c3[gateNeg(C,A)] || c1[gateNeg(A,B)] || c2[gateNeg(B,C)]
| A *-> c2[A] | c1[A] *-> A
| B *-> c3[B] | c2[B] *-> B
| C *-> c1[C] | c3[C] *-> C
| A ~ B + C *-> B-C

```

This model then yields the same solutions as for the original repressilator, but the additional reactions now appear in the resulting LBS model. If all reactions are intended to be used for simulation only, the star can be omitted if the "Simulation-only" box under the editor is checked. In the general form of a reaction, the catalyst, reactants and/or products may be omitted.

Quantitative constraints

Additional quantitative constraints can also be imposed on a model. For example, the following specifies any ribosome binding site for which the rate of translation is greater than 0.05:

```
rbs<rate(R)> | R > 0.05
```

Rate variables in modules should typically be combined with the *new variable* operator to ensure that different instances of a module may have different numbers assigned to the variable, subject to the quantitative constraints. For example, a constraint on the translation rate in the repressilator module can be written as follows:

```

module gateNeg(i,o) {
  new R.
  prom<neg(i)>; rbs<rate(R)>; pcr<codes(o)>; ter
  | R > 0.01
};

```

The syntax of the Visual GEC language

This section defines the concrete syntax of the Visual GEC language in terms of a context-free grammar. The grammar relies on the following lexical conventions, where we write “digit” for a single character in the range 0-9, and “alphanumeric” for any character in the range A-Z or a-z.

- *Integer*: a non-empty sequence of digits.
- *Name*: the first character of a name must be a lower-case alphanumeric. This is followed by a possibly-empty sequence of characters which may be alphanumeric or digits.
- *Variable*: the first character of a name must be an upper-case alphanumeric. This is followed by a possibly-empty sequence of characters which may be alphanumeric or digits.
- *Comma-separated lists (possibly empty)* are written with an underline.
- *Float*: there are three different ways to produce a float value:
 1. One or more digits followed by a decimal point (.), followed by zero or more digits. For example: 3.141.
 2. One or more digits followed by an uppercase ‘E’ or lowercase ‘e’, followed by a plus (+) or minus (-) sign, followed by one or more digits. For example: 3e-5.
 3. One or more digits followed by a decimal point, followed by zero or more digits, followed by an uppercase ‘E’ or lowercase ‘e’, followed by a plus or minus sign, followed by one or more digits. For example: 1.4324e+2.

Single-line comments are prefixed with “//”. Multi-line comments are opened with (* and closed with *), and may be nested.

The grammar for Visual GEC is then defined as follows, where terminal symbols of are written in teletype font and non-terminals are in **bold**.

Non-terminal	Definition	Description
VGEC ::=	<u>Directive</u> <i>GEC</i>	A Visual GEC program
Directive ::=	<i>directive</i> <i>sample</i> Float	End time for simulation
	<i>directive</i> <i>sample</i> Float Integer	- with optional data points
	<i>directive</i> <i>scale</i> Float	Scaling factor
	<i>directive</i> <i>time</i> TU	Unit of time
	<i>directive</i> <i>concentration</i> CU	Unit of concentration
	<i>directive</i> <i>plot</i> Plots	Species to plot
	<i>directive</i> <i>tolerance</i> Float	ODE solver tolerance
Plots ::=	PlotSpec	One species to plot
	PlotSpec ; Plots	- or more
SimpSpec ::=	Name	Simple species
	Name [Name]	- in compartment
PlotSpec ::=	SimpSpec	Atomic plot species
	SimpSpec - PlotSpec	Complex plot species

GEC ::=	Apart	Abstract part
	GEC ; GEC	Sequential composition
	GEC GEC	Parallel composition
	Name [GEC]	Compartment
	GEC Constraint	Constraint composition
	new Name . GEC	New variable
	module Name (FPar) {GEC}; GEC	Module definition
	Name(APar)	Module invocation
APart ::=	AName : PartType <Property>	Asbstract part
	PartType <Property>	- without name
	AName : PartType	- without properties
	PartType	- without name & properties
PartType ::=	Prom	Promoter type
	Rbs	Ribosome binding site type
	Pcr	Protein coding region type
	Ter	Terminator type
Property ::=	pos (AComplex)	Positive regulation
	pos (AComplex, AFloat , AFloat, AFloat)	- with rates
	neg (AbsSpec)	Negative regulation
	neg (AbsSpec, AFloat, AFloat, AFloat)	- with rates
	con (AFloat)	Constitutive expression
	rate (AFloat)	Translation rate
AFloat ::=	Float	Concrete abstract float
	Variable	Variable abstract float
	_	Wild card
AName ::=	Name	Concrete abstract name
	Variable	Variable abstract name
	_	Wild card
ASpec ::=	AName	Atomic abstract species
	AName – Acomplex	Complex abstract species
Constraints ::=	Reaction	Reaction constraint
	Transport	Transport constraint
	NumCons	Numerical constraint
	Constraint Constraint	Constraint composition
Reaction ::=	ASpec ~ Sum Arrow Sum	Enzymatic reaction
	Sum ->{AFloat} Sum	Standard reaction
Sum ::=	ASpec	Atomic sum
	ASpec + Sum	Composite sum
Arrow ::=	->	Simple arrow
	*->	- simulation-only

	\rightarrow { AFloat }	Arrow with rate
	$\ast\rightarrow$ { AFloat }	- simulation-only
Transport ::=	ASpec Arrow Name [ASpec]	Transport into compartment
	Name [ASpec] Arrow ASpec	Transport out of compartment
NumCons ::=	ASpec > ASpec	Greater than constraint
TU ::=	seconds s minutes m hours h	Units of time
CU ::=	molar M	1 mol·L ⁻¹
	milimolar mM	10 ⁻³ mol·L ⁻¹
	micromolar uM	10 ⁻⁶ mol·L ⁻¹
	nanomolar nM	10 ⁻⁹ mol·L ⁻¹
	picomolar pM	10 ⁻¹² mol·L ⁻¹
	femtomolar fM	10 ⁻¹⁵ mol·L ⁻¹
	attomolar aM	10 ⁻¹⁸ mol·L ⁻¹
	zeptomolar zM	10 ⁻²¹ mol·L ⁻¹
	yoctomolar yM	10 ⁻²⁴ mol·L ⁻¹

Known bugs

The following bugs are known for the currently released version of Visual GEC:

- In the Reaction Graph tab, zoom sometimes does not respond to selection.
- The **plot** directive is sensitive to the ordering of species in a complex.