BioNetGen Quick Reference Guide

Description

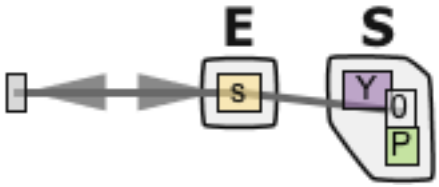
BioNetGen (bionetgen.org) is software for the specification and simulation of rule-based models of biochemical systems, including signal transduction, metabolic, and genetic regulatory networks. The rule-based approach allows for the maintenance of detailed information on molecular structures and interactions, as well as significant scalability both in model construction and simulation.

Installation

1. Download RuleBender, a Graphical User Interface for BioNetGen (rulebender.org).
2. Java: If needed, install Java version 1.6 or greater from the Oracle download site (oracle.com/java).
3. Perl: Perl is installed on most Unix-like operating systems. You may need to install it on Windows. We recommend ActivePerl.

Required Model Components

Block Name	*Description*
`parameters`	Values and expressions for parameters in the model.
`molecule types`	Molecule definitions including components and component states.
`seed species`	Initial conditions for molecules and complexes present at simulation start time.
`observables`	Sums over concentrations of species with properties specified using patterns.
`functions`	Functions of observables used to construct non-elementary rate laws that depend on global or local properties.
`reaction rules`	Generate reactions with the specified rate law based on selecting and transforming reactants using patterns.
`actions`	Sequence of actions used to simulate the model.



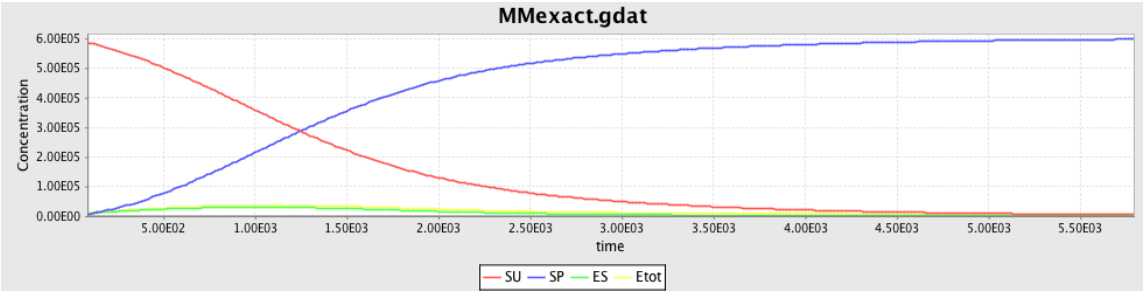
Contact Map

Shows the molecules, components, component states and bonds, as well as synthesis and deletion of molecules. Right-clicking on molecules allows the user to search for similarly named proteins in a number of established databases.

Example BNGL Model

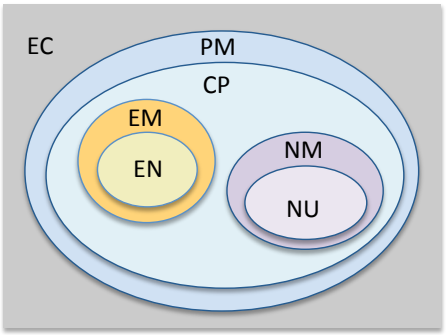
```
# MM Example
begin model
begin parameters
    # Avogadro's number - scaled for umol
    NA   6.022e23/1e6
    # Cell volume
    V    1e-12 # liters - typical for eukaryote
    # Rate constants
    kp1  1.0/(NA*V) # 1/uM 1/s -> 1/molecules 1/s
    km1  0.1 # 1/s
    k2   0.01 # 1/s
    ksyn 1e-4*(NA*V) # uM/s -> molecules / s
    kdeg 0.01 # 1/s
    # Initial concentrations
    E0   0.01*NA*V # uM -> molecules
    S0   1.0*NA*V  # uM -> molecules
end parameters
begin molecule types
    E(s)
    S(Y~0~P)
end molecule types
begin seed species
    E(s)   E0
    S(Y~0) S0
end seed species
begin observables
    Molecules SU S(Y~0)
    Molecules SP S(Y~P)
    Molecules ES E(s!1).S(Y!1)
    Molecules Etot E()
end observables
begin reaction rules
    ESbind: E(s) + S(Y~0) <-> E(s!1).S(Y~0!1) kp1,km1
    ESconvert: E(s!1).S(Y~0!1) -> E(s) + S(Y~P) k2
    Esyndeg: 0 <-> E(s) ksyn, kdeg
end reaction rules
end model
# actions
generate_network({overwrite=>1})
visualize({type=>"regulatory",groups=>1,\
           collapse=>1,opts=>"opts.txt"})
simulate({method=>"ode",t_end=>20000,n_steps=>1000})
```

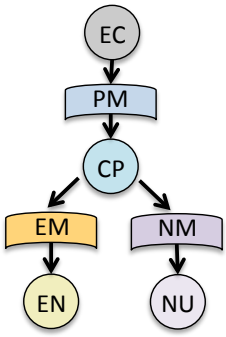
Simulation Results



Compartmental BioNetGen (bionetgen.org/index.php/Compartments_in_BNGL)
Extension of BNGL to enable explicit modeling of the compartmental organization of the cell and its effects on system dynamics. Introduces localization attributes for both molecules and species, as well as appropriate volumetric scaling of reaction rates.

```
begin compartments
    EC  3  vol_EC                # extracellular space
    PM  2  sa_PM*eff_width  EC   # plasma membrane
    CP  3  vol_CP           PM   # cytoplasm
    NM  2  sa_NM*eff_width  CP   # nuclear membrane
    NU  3  vol_NU           NM   # nuclear space
    EM  2  sa_EM*eff_width  CP   # endosomal membrane
    EN  3  vol_EN           EM   # endosomal space
end compartments
```





Context and Pattern Matching within Rules
Context (shown below in red) encompasses components, states, and bonds in a rule that do not undergo transformation.

Reaction Rule	*Reactant Requirements*	*Reactant Species Matched*
`R1: R(l!+,Y~0) -> R(l!+,Y~P) k1`	R must be unphosphorylated at Y and bound at l (binding site for L).	`R(l!1,Y~0).L(r!1)`
`R2: R(l,Y~0) -> R(l,Y~P) k2`	R must be unphosphorylated at Y and *not* bound at l.	`R(l,Y~0)`
`R3: R(Y~0) -> R(Y~P) k3`	R must be unphosphorylated at Y.	`R(l!1,Y~0).L(r!1)` `R(l,Y~0)`

Context restricts the application of a given rule. Rules with more context are more specific and rules with less context are more general. Using more context allows for greater control over rates of specific reactions at the cost of requiring more rules to specify a model. Here, R1 and R2 specify different phosphorylation rates depending on whether R is bound at l, whereas R3 specifies a rate that is independent of the state of l.

Functional Rate Laws (bionetgen.org/index.php/Functions)
A function is a mathematical expression that can involve numbers, parameters, observables, and other pre-defined functions and operators (see bionetgen.org/index.php/Built-ins). Functions can be used to define reaction rules that do not obey mass-action kinetics. Importantly, functions *substitute* for rate constants, meaning that the rate of a reaction is calculated as the product of the function and the reactant species concentrations (analogous to mass-action rate laws). Functions can be defined "inline" following a reaction rule or within a separate block of the BNGL file, with the following syntax (`[...]` denotes an optional argument):

```
begin functions
   [label:]  func_name([arg]) [=] math_expression
      ⋮
end functions
```

Here, `arg` is a "pointer" to a specific complex or molecule and limits the scope over which observables within the function definition are calculated (the default scope is the entire system). Functions that do not accept an argument are termed "global functions" and those that do are termed "local functions." Pointers to complexes and molecules are specified by "tagging" a reactant pattern in a rule using the `%tag` syntax, where `tag` can be any alphanumeric string of characters. Tags prefixed to a pattern (e.g., `%x:A()`) point to the *complex* matched by the pattern and tags postfixed (e.g., `A()%x`) point to the matched *molecule*.

Examples:

```
begin observables
   Molecules  Atot  A()                  # Observable counting total number of A molecules
   Molecules  Bp    B(c~P)               # Observable counting number of phosphorylated B molecules
end observables
begin functions
   gfunc() = 0.5*Atot^2/(10+Atot^2)        # Hill function (global) defined over all A molecules in the system
   lfunc(x) = 0.5*Atot(x)^2/(10+Atot(x)^2) # Hill function (local) defined over all A molecules in "x"
end functions
begin reaction rules
   B(c) + C(b) -> B(c!1).C(b!1) gfunc()          # Binding rate depends on no. of A molecules in the system
   B(c) + C(b) -> B(c!1).C(b!1) 0.5*Atot^2/(10+Atot^2)  # Ditto, but with "inline" function definition
   %y:B(c) + C(b) -> %y:B(c!1).C(b!1) lfunc(y)    # Prefix tag limits observable scope to complex containing B
   B(c)%y + C(b) -> B(c!1)%y.C(b!1) Bp(y)*gfunc() # Postfix tag limits observable scope to B molecule only
end reaction rules
```

Maintaining Consistent Units

Units are not enforced within BioNetGen, but for realistic results units among all concentrations and rate constants must be kept consistent. The most common units utilized are in terms of molecules/cell. (N_A: Avogadro's number; V: volume)

Common Parameters	*Example Starting Value*	*Desired Units*	*Conversion Factor*	*Final Parameter*
Volumes	10^{-6} μl	l	10^{-6} l/μl	10^{-12} l
Concentrations	0.01 μM	molec	$N_A \times V/10^6$ molec/μM	6022 molec
1st-order rate constants	1 s^{-1}	s^{-1}	*None*	1 s^{-1}
2nd-order rate constants	10^5 $M^{-1}s^{-1}$	$molec^{-1}s^{-1}$	$1/(N_A \times V)$ M/molec	1.66×10^{-7} $molec^{-1}s^{-1}$

Reading Models from File

In addition to `.bngl` files, BioNetGen can read in pre-generated reaction networks from `.net` files and SBML models with the `.xml` extension. This is done using the `readFile({file=>"filename"})` action. Optional arguments include `blocks=>["blockname1", "blockname2",...]` and `atomize=>0/1`. If the filename ends with `.xml`, the SBML-to-BNGL translator is automatically called. The `atomize=>1` option invokes the "Atomizer," a method for extracting implicit molecular structure from flat SBML species. The Atomizer can also be accessed as a standalone web application at ratomizer.appspot.com/translate.

Simulation Methods

Method	*Command*	*Description*
Ordinary differential equations	`simulate({method=>"ode"})`	Network-based deterministic simulation using the SUNDIALS CVODE ODE solver.
Stochastic simulation algorithm	`simulate({method=>"ssa"})`	Network-based stochastic simulation using the Gillespie direct method to sample over reactions.
Partitioned-leaping algorithm	`simulate({method=>"pla"})`	Network-based stochastic simulation using a tau-leaping variant to speed up simulation.
Network-free simulation	`simulate({method=>"nf"})`	Stochastic simulation that does not require network generation. Uses the Gillespie direct method to sample over reaction rules. Scalable to models encoding large reaction networks (see nfsim.org for more information).

Parameter Scans

BioNetGen allows the user to run parameter scans, which quantify the effects of varying the value of an individual parameter on any observable species. This is done using the command `parameter_scan({argument=>value})` with the following arguments:

Argument	*Usage*	*Example*
`parameter`	Name of the parameter to be varied.	`parameter=>"kp1"`
`par_min`	Minimum value the parameter will take.	`par_min=>1e-3`
`par_max`	Maximum value the parameter will take.	`par_max=>1e3`
`n_scan_pts`	Number of values the parameter will take, uniform between `par_min` and `par_max`.	`n_scan_pts=>1000`
`log_scale`	Selects parameter values uniformly between $\log_{10}$(`par_min`) and $\log_{10}$(`par_max`).	`log_scale=>1`
`method`	Simulation method to be used (`"ode"`, `"ssa"`, `"pla"`, or `"nf"`).	`method=>"ode"`
`t_end`	Simulation run time.	`t_end=>10000`
`n_steps`	Number of observable outputs.	`n_steps=>1000`

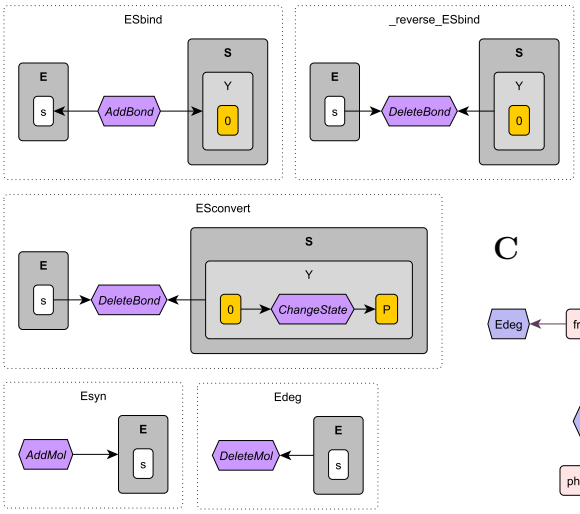
Exporting Models to Other Formats

Action	*File Extension(s)*	*Description*
`writeModel`	.bngl	BNGL model file.
`writeNetwork`	.net	Reaction network file used for network-based simulations.
`writeSBML`	.xml	Systems Biology Markup Language; compatible with many software packages.
`writeMfile`	.m	Model file compatible with Matlab.
`writeMexfile`	.cvode.c, .m	Model file written in C that can be compiled into a Matlab executable (MEX) file.
`writeXML`	.xml	Model file usable for network-free simulation in NFsim.
`writeMDL`	.mdl, .py, .geometry.mdl	Model file usable for spatial simulations in MCell.
`visualize`	.gml	Graph Modeling Language file for visualizations.

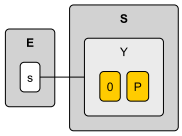
Energy Modeling in BioNetGen (http://bionetgen.org/index.php/Energy_Modeling)
Extension of BNGL to enable definition of energy patterns to drive model kinetics based on changes in free energy of reactions. Typically allows for the writing of fewer rules for each model, less context in each rule, and the generation of reaction networks guaranteed to satisfy detailed balance.

Visualization (bionetgen.org/index.php/Visualization)
Visualization tools are accessed using one or more `visualize({type=>"string"})` commands. Optional arguments include `each=>1` to generate a separate output file for each rule and `suffix=>"string"` to append a suffix to the file name. The output is a Graph Modeling Language file `[model]_[type]_[suffix].gml`, which can be processed by graph layout software such as yEd (yworks.com/yed).

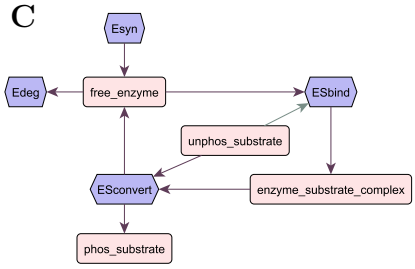
A



B



C



D

opts.txt

```
begin background
      begin include
            E
            _reverse_ESbind
      end include
      begin exclude
            E(s)
            S(Y~0)
      end exclude
end background
begin classes
      begin free_enzyme
            E(s)
      end free_enzyme
      begin unphos_substrate
            S(Y~0)
      end unphos_substrate
      begin phos_substrate
            S(Y~P)
      end phos_substrate
      begin enzyme_substrate_complex
            E(s!1).S(Y!1))
      end enzyme_substrate_complex
end classes
```

A. Rule visualization: Rules can be visualized either in terms of graph operations (`type=>"ruleviz_operation"`) or reactant and product patterns (`type=>"ruleviz_pattern"`).
Command used here: `visualize({type=>"ruleviz_operation"})`.

B. Contact map: Equivalent to the contact maps generated by RuleBender (except for the synthesis/degradation directed edges).
Command used here: `visualize({type=>"contactmap"})`.

C. Regulatory graph: Graphical representation of the basic processes in a model. Here, processes are grouped (`groups=>1`) and group nodes are collapsed (`collapse=>1`) into single nodes. Additional options exist for displaying hidden sites and processes (`background=>1`) and for outputting the visualization to a human-readable text file (`textonly=>1`).
Command used here: `visualize({type=>"regulatory",groups=>1,collapse=>1,opts=>"opts.txt"})`.

D. User options: For regulatory graphs, user-defined commands can be specified in text files and loaded using `opts=>"filename"` or `opts=>["filename1","filename2",...]`. Useful for modifying the background and providing intuitive names to groups of sites.