Chapter 5

Rule-Based Modeling of Biochemical Systems with BioNetGen

James R. Faeder, Michael L. Blinov, and William S. Hlavacek

Summary

Rule-based modeling involves the representation of molecules as structured objects and molecular interactions as rules for transforming the attributes of these objects. The approach is notable in that it allows one to systematically incorporate site-specific details about protein–protein interactions into a model for the dynamics of a signal-transduction system, but the method has other applications as well, such as following the fates of individual carbon atoms in metabolic reactions. The consequences of protein–protein interactions are difficult to specify and track with a conventional modeling approach because of the large number of protein phosphoforms and protein complexes that these interactions potentially generate. Here, we focus on how a rule-based model is specified in the BioNetGen language (BNGL) and how a model specification is analyzed using the BioNetGen software tool. We also discuss new developments in rule-based modeling that should enable the construction and analyses of comprehensive models for signal transduction pathways and similarly large-scale models for other biochemical systems.

Key words: Computational systems biology, Mathematical modeling, Combinatorial complexity, Software, Formal languages, Stochastic simulation, Ordinary differential equations, Protein–protein interactions, Signal transduction, Metabolic networks.

1. Introduction

BioNetGen is a set of software tools for rule-based modeling *(1)*. Basic concepts of rule-based modeling and the BioNetGen Language (BNGL) are illustrated in **Fig. 1** – these concepts and the conventions of BNGL will be thoroughly discussed later in the text. Here, in explaining how to use BioNetGen to model biochemical systems, we will be primarily concerned with signal-transduction systems, which govern cellular responses, such as growth and differentiation, to signals, such as hormones and cytokines. In other

Ivan V. Maly (ed.) *Methods in Molecular Biology, Systems Biology, vol. 500*

DOI: 10.1007/978-1-59745-525-1_5

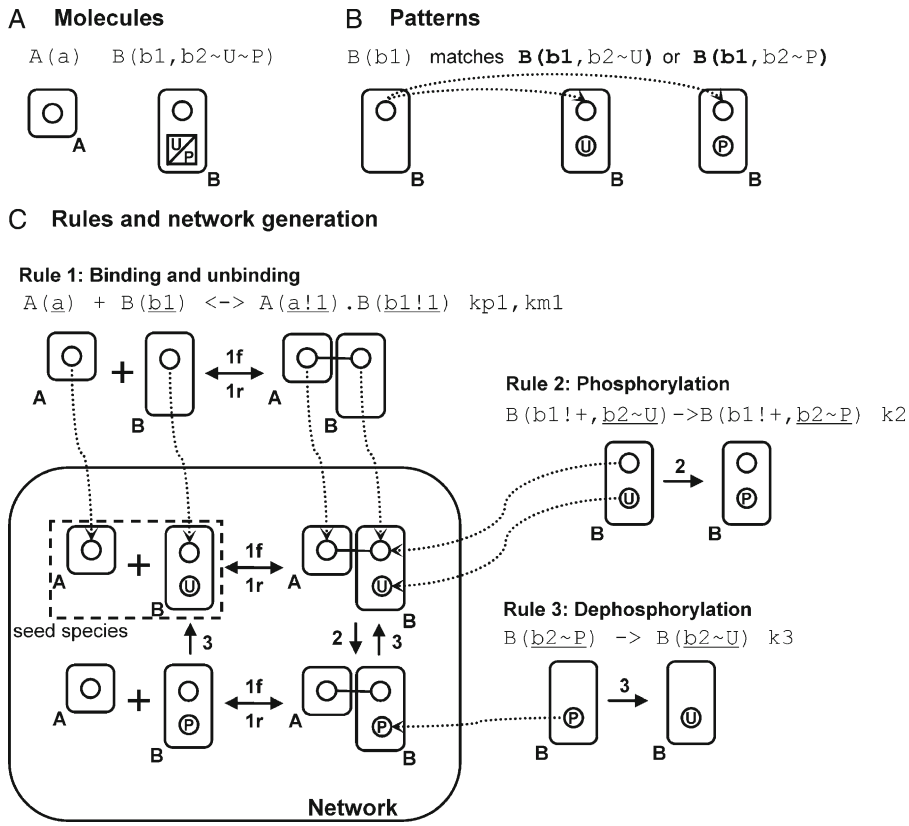

Fig. 1. Rule-based modeling concepts and their encoding in BioNetGen Language (BNGL). (**A**) The basic building blocks are molecules, which are structured objects, composed of components that represent functional elements of proteins and may have associated states that represent covalent modifications or conformations. Molecules may be assembled into complexes through bonds that link components of different molecules. (**B**) Patterns select particular attributes of molecules in species (shown in *bold*). The pattern shown here selects molecules of B with a free b1 binding site regardless of the phosphorylation or binding status of the b2 component. (**C**) Rules specify the biochemical transformations that can take place in the system and may be used to build up a network of species and reactions (*see* **Section 3.5** for a complete description of rule syntax). The reaction center (components undergoing direct modification) is underlined. (This is shown for clarity and is not part of BioNetGen syntax.) Starting with the seed species, rules are applied to generate new reactions and species by mapping reactant patterns onto species and applying the specified transformation(s). Species generated by new reactions may be acted on by other rules to generate new reactions and species, and the process continues until no new reactions are found or some other stopping criteria are satisfied.

words, signal-transduction systems are responsible for making decisions about the fates and activities of cells. Decision making in these systems is accomplished by dynamical systems of interacting molecules *(2)*. To develop predictive computational models of these complex systems, we must be able to abstract their relevant details in a form that enables reasoning about or simulation of the logical consequences of a set of interactions, which enables the testing of model predictions against experimental observations *(3)*. Analysis of predictive models can help to guide experimental

investigations and may ultimately enable model-guided engineering and manipulation of cellular regulation *(4–6)*. Before beginning our discussion of BioNetGen, we will briefly recap features of signal-transduction systems that motivate a rule-based modeling approach and the general idea of rule-based modeling. For more thorough reviews of these topics see **refs.** *7, 8*.

A prominent feature of any signal-transduction system is an intricate network of protein–protein interactions *(9, 10)*. These interactions can have a number of consequences, including the posttranslational modification of proteins, the formation of heterogeneous protein complexes in which enzymes and substrates are colocalized, and the targeted degradation of proteins. For understanding and modeling the system dynamics of protein–protein interactions, the details that are most relevant are typically found at the level of protein sites, the parts of proteins that are responsible for protein–protein interactions. These interactions are mediated by evolutionarily conserved modular domains of proteins that have binding and catalytic activities, such as Src homology 2 (SH2) domains and protein tyrosine kinase domains, and by short linear motifs (e.g., immunoreceptor tyrosine-based activation motifs or ITAMs) *(11)* with binding activities that can often be switched on and off through posttranslational modifications, such as tyrosine phosphorylation *(12–14)*. A great deal of knowledge about the site-specific details of protein–protein interactions has accumulated in the scientific literature and is being actively organized in electronic databases *(15, 16)*, and new technologies, such as mass spectrometry (MS)-based proteomics *(17)*, can be applied to quantitatively monitor system responses to a signal at the level of protein sites on a large scale. For example, time-resolved measurements of the phosphorylation of individual tyrosine residues are possible *(18)*.

Despite the high relevance of the site-specific details of protein–protein interactions for understanding system behavior, models incorporating these details are uncommon. For example, the seminal model of Kholodenko et al. *(19)* and many of its extensions, such as the model of Schoeberl et al. *(20)*, for early events in signaling by the epidermal growth factor receptor (EGFR) do not track the phosphorylation kinetics of individual tyrosines in EGFR. Models that incorporate such details are generally difficult or impossible to specify and analyze using conventional methods, largely because of the combinatorial number of protein modifications and protein complexes that can be generated through protein–protein interactions *(7, 8)*. For example, a protein containing n peptide substrates of kinases can potentially be found in up to 2^n distinct phosphorylation states. This feature of protein–protein interactions, which arises because a typical protein involved in cellular regulation contains multiple sites of posttranslational modification and multiple binding sites, has

been called combinatorial complexity and has been recognized as a significant challenge to our understanding of cellular regulation *(7, 21, 22)*. In a conventional model specification, which often takes the form of a list of the reactions that are possible in a signal-transduction system or the corresponding system of coupled ordinary differential equations (ODEs) for the chemical kinetics, each chemical species that can be populated and each reaction that can occur must be manually defined, which is infeasible for all but the simplest systems because of the vast numbers of chemical species and reactions that can usually be generated by protein–protein interactions.

Another limitation of conventional modeling is a lack of standards for explicitly representing the composition and connectivity of molecular complexes. The chemical species accounted for in a typical model are represented as structureless objects whose identities and properties are referenced only by name. Modelers attempt to name model parameters and variables such that their names suggest what is being represented, but conventions vary and are often inconsistent. A dimer of EGFR molecules may be represented as R-R or R:R – designations that abbreviate EGFR to R (for receptor) and that indicate the composition of the complex – or simply as D (for dimer). A dimer of EGFR molecules associated with the adapter protein Grb2 may then be represented as R-R-Grb2, D-Grb2, or even as R-Grb2 or simply by the index of a generic variable name (e.g., X_5). The latter examples obscure the fact that two receptor molecules are present in the complex. Model(er)-specific nomenclatures thus present a challenge to understanding a model, especially a large model, which becomes particularly problematic when one attempts to reuse or extend a model. In addition, information about how two molecules are connected is nearly always absent in a conventional model specification, even though in many cases there is detailed site-specific information available about the interaction. For example, interaction of EGFR and Grb2 occurs when the SH2 domain of Grb2 binds a phosphorylated tyrosine residue in EGFR, such as Y1068 *(23)*.

The limitations of conventional approaches to model specification noted above have prompted the development of formal languages specially designed for representing proteins and protein–protein interactions, the κ-calculus being an early and notable example *(24)*. One of these formal languages is the BioNetGen language (BNGL) *(8)*, which is based on the use of graphs to represent proteins and protein complexes and graph-rewriting rules to represent protein–protein interactions *(25, 26)*. BNGL allows site-specific details of protein–protein interactions to be captured in models for the dynamics of these interactions in a systematic fashion, alleviating both nomenclature and reusability issues *(8)*. BNGL also provides a means for specifying precise visualizations of protein–protein interactions *(25, 26)*. Below,

we provide a thorough overview of the text-based syntax and semantics of BNGL, an understanding of which is essential for using the BioNetGen software (http://bionetgen.org). BioNetGen facilitates a rule-based approach to modeling biochemical reaction kinetics, an alternative to conventional modeling that largely overcomes the problem of combinatorial complexity *(8)*. We note that the current syntactical and semantic conventions of the κ-calculus are nearly identical to those of BNGL *(27)*.

In a rule-based approach to modeling, the molecular interactions in a system are abstracted as BNGL-encoded rules, which are precise formal statements about the conditions under which interactions occur and the consequences of these interactions. Rules also provide rate laws for transformations resulting from molecular interactions. At one extreme, a rule simply corresponds to an individual chemical reaction. However, a rule is far more useful when local context governs an interaction, and the rule can be specified such that it defines not a single reaction but a potentially large class of reactions, all involving a common transformation parameterized by the same rate law. The use of such rules to model protein–protein interactions can often be justified, at least to a first approximation, by the modularity of proteins *(12)*. Rules can be used to obtain predictions about a system's behavior in multiple ways. For example, they can serve as generators of a list of reactions. In other words, a set of rules, which can be viewed as a high-level compact definition of a chemical reaction network, can be used to obtain a conventional model specification *(1, 28, 29)*, which can then be analyzed using standard methods. Alternatively, rules can serve as generators of discrete reaction events in a kinetic Monte Carlo simulation of chemical kinetics *(21, 30, 31)*. A rule-based model is capable of comprehensively accounting for the consequences of protein–protein interactions, including all possible phosphoforms of a protein and the full spectrum of possible protein complexes implied by a given set of interactions. Such a model is specified using BNGL in a BioNetGen input file, which may also contain directions for processing the model specification. For example, actions may be defined for simulating a model and producing desired outputs. In the following, we will describe the elements of an example input file in detail.

Since our initial application of a rule-based modeling approach in 2001 to study signaling by the high-affinity IgE receptor *(32–34)*, the software that we have used in our work – initially a FORTRAN code called EQGEN – has evolved dramatically and has been applied to study a number of other biochemical systems *(35–39)*. The initial version of BioNetGen was released in 2004 *(1)*. The name "BioNetGen" is a mnemonic for "Biological Network Generator," but this name should not be interpreted to delimit the full range of the software's capabilities. The software not only

generates reaction networks from rules, but also simulates such networks using a variety of methods. Iterative application of rules to a set of seed species (*see* **Fig. 1c**) may be used to generate a network in advance of a simulation, which may subsequently be carried out either by numerically solving ODEs or by implementing a stochastic simulation algorithm (SSA) *(40–42)*. Alternatively, rules may be applied during a simulation as the set of populated species grows, a procedure that has been called "on-the-fly" network generation and simulation *(28, 29)*. Finally, network generation may be avoided altogether by instantiating individual instances of chemical species and carrying out a discrete-event particle-based simulation, in which rules serve as event generators *(21, 30, 31)* (*see* **Subheading 3.7.2**). Simulation engines implementing such methods will soon be available within the BioNetGen framework and will be called through interfaces similar to those of the existing engines (*see* **Subheading 3.6**).

Later, we summarize essentially everything a modeler needs to know to start developing and analyzing rule-based models with BioNetGen. After an overview of the BioNetGen software distribution, we present a step-by-step guide to writing a BioNetGen input file, in which we carefully explain the elements of an example input file. Numerous tips and tricks can be found in the **Notes** section. Building on the basics, we then present several examples that illustrate more advanced BioNetGen capabilities. Finally, we briefly discuss new developments in rule-based modeling that should enable the construction and analyses of large-scale comprehensive models for signal-transduction systems.

2. Software

BioNetGen is a set of integrated open-source tools for rule-based modeling. A schematic of the software architecture is shown in **Fig. 2**. The software and documentation are available at http://bionetgen.org, a wiki site. Downloading the software or modifying the wiki pages requires user registration with a valid email address. The software is easy to install and runs with no compilation on Linux, Mac OS X, and Windows operating systems (*see* **Note 1**). BioNetGen can be also used online (without installation) from within the Virtual Cell modeling environment (http://vcell.org/bionetgen).

The components of BioNetGen include the network generation engine BNG2, which is written in Perl, the simulation program Network, which is written in C, a plotting program called PhiBPlot, which is written in Java, and a graphical front-end called RuleBuilder, which is also written in Java. The core

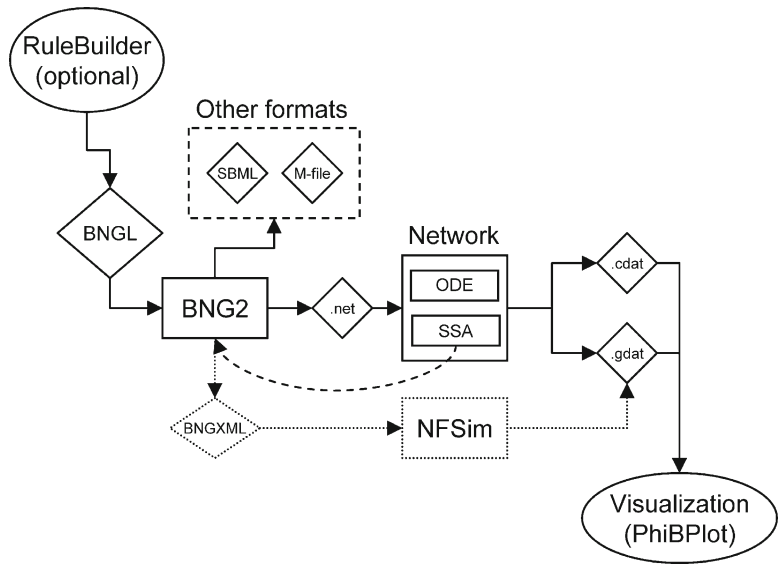

Fig. 2. Software architecture of BioNetGen. The BioNetGen language (BNGL) file specifies a rule-based model that can be processed by the BioNetGen core version 2 (BNG2) in a variety of ways. Iterative application of rules to an initial set of species can generate a reaction network that is passed to one of the simulation modules through the .net format or exported to formats (SBML, MATLAB) that can be read by other programs. In the near future, an XML-based encoding will be used to pass model specifications among additional software components, including a particle-based simulator called NFsim ("network-free" simulator) (Sneddon, M., Faeder, J. and Emonet, T., private communication). Simulation modules produce .cdat and .gdat files, which record the time courses of species concentrations and observables, respectively. The *dashed arrow* connecting the SSA module and BNG2 represents the on-the-fly network generation capability available for stochastic simulations.

component, BNG2, which has a command-line interface, processes BioNetGen input files to generate two kinds of outputs: a chemical reaction network derived by processing rules and/or the results of simulating a model (*see* **Note 2**). Input files are discussed below at length. Reaction networks are exported in a native .net format, in M-file format for processing by MATLAB (The MathWorks, Natick, MA), and in Systems Biology Markup Language (SBML), which is a community-developed standard for the encoding of biological models *(43)*. A network encoded in SBML can be processed by a variety of SBML-compliant software tools (for a list of these tools, see http://sbml.org). An example of an SBML-compliant tool that complements BioNetGen is COPASI *(44)*, which provides model analysis capabilities, such as parameter estimation methods, unavailable in the native BioNetGen environment. Simulation results are exported as tabular data in plain-text files that have the extension .cdat or .gdat. A .cdat file contains time series for concentrations of chemical species. A

.gdat file contains time series for observables defined in a BioNetGen input file or .net file (*see* **Subheading 3.3**). Simulations specified in an input file are preprocessed by BNG2 and then passed to Network, which is a simulation engine driver. Network interfaces with the CVODE package *(45, 46)*, a set of routines for solving stiff and nonstiff initial value problems for systems of ODEs. Network also provides an implementation of the direct method of Gillespie *(40)* for stochastic simulations. The command-line interface of Network allows a .net file to be processed directly without preprocessing by BNG2, but this option is unavailable for simulation in on-the-fly mode *(28, 29)*, which necessarily requires communication between BNG2 and Network. On-the-fly simulation is discussed further in **Subheading 3.7.2**. PhiBPlot is a utility for producing *x*–*y* plots from .cdat and .gdat files. The .cdat and .gdat files can also be processed by other plotting tools, such as Grace (http://plasma-gate.weizmann.ac.il/Grace). RuleBuilder provides a graphical user interface to BioNetGen. It also provides a drawing tool for creating and editing models that may be particularly helpful to new users.

BioNetGen has been integrated into the Virtual Cell (VCell) modeling environment (http://vcell.org) as a stand-alone application called BioNetGen@VCell. A BioNetGen service is callable from a VCell user interface and runs on a client computer. The VCell user interface can be used to visualize and export simulation results. Alternatively, a VCell BioModel can be automatically created from an SBML file generated by BioNetGen@VCell.

3. Methods

We will illustrate the method of constructing a rule-based model by stepping through the BioNetGen input file shown in **Listing 1**, which specifies a simplified version of a model for early events in EGFR signaling *(35)*. Additional examples can be found in the `Models2` directory of the BioNetGen distribution available from http://bionetgen.org, or on the Web at http://vcell.org/bionetgen/samples.html. A BioNetGen input file contains the information required to specify a model, including definitions of molecules, rules for molecular interactions, and model outputs, which we call "observables." An input file may also contain commands called "actions" that act on the model specification, such as generating the network of species and reactions implied by rules, performing simulations, and translating the model into other formats. The syntax of actions is borrowed from the Perl programming language. Model elements are specified in blocks delimited by "`begin`" and "`end`" tags as indicated in **Listing 1**.

Listing 1. Elements of the BioNetGen input file `egfr_simple.bngl`. Block names are shown in **bold**, and reaction centers are underlined for clarity in the `reaction rules` block.

```
begin parameters
  NA 6.02e23 # Avogadro's number (molecues/mol)
  f  1       # Fraction of the cell to simulate
  Vo f*1.0e-10 # Extracellular volume=1/cell_density (L)
  V  f*3.0e-12 # Cytoplasmic volume (L)
  # Inital amount of ligand (20 nM)
  EGF_init 20*1e-9*NA*Vo # convert to copies per cell
  # Initial amounts of cellular components (copies per cell)
  EGFR_init     f*1.8e5
  Grb2_init     f*1.5e5
  Sos1_init     f*6.2e4
  # Rate constants
  # Divide by NA*V to convert bimolecular rate constants
  # from /M/sec to /(molecule/cell)/sec
  kp1 9.0e7/(NA*Vo) # ligand-monomer binding
  km1 0.06          # ligand-monomer dissociation
  kp2 1.0e7/(NA*V) # aggregation of bound monomers
  km2 0.1           # dissociation of bound monomers
  kp3 0.5           # dimer transphosphorylation
  km3 4.505         # dimer dephosphorylation
  kp4 1.5e6/(NA*V) # binding of Grb2 to receptor
  km4 0.05          # dissociation of Grb2 from receptor
  kp5 1.0e7/(NA*V) # binding of Grb2 to Sos1
  km5 0.06          # dissociation of Grb2 from Sos1
  deg 0.01          # degradation of receptor dimers
end parameters

begin molecule types
  EGF(R)
  EGFR(L,CR1,Y1068~U~P)
  Grb2(SH2,SH3)
  Sos1(PxxP)
  Trash()
end  molecule types

begin seed species
  EGF(R)               0
  EGFR(L,CR1,Y1068~U) EGFR_init
  Grb2(SH2,SH3)       Grb2_init
  Sos1(PxxP)          Sos1_init
end seed species

begin observables
  1 Molecules  EGFR_tot  EGFR()
  2 Molecules  Lig_free  EGF(R)
  3 Species    Dim       EGFR(CR1!+)
```

```
  4 Molecules  RP EGFR(Y1068~P!?)
  # Cytosolic Grb2-Sos1
  5 Molecules  Grb2Sos1  Grb2(SH2,SH3!1).Sos1(PxxP!1)
  6 Molecules  Sos1_act
EGFR(Y1068!1).Grb2(SH2!1,SH3!2).Sos1(PxxP!2)
end observables

begin reaction rules
  # Ligand-receptor binding
  1 EGFR(L,CR1) + EGF(R) <-> EGFR(L!1,CR1).EGF(R!1) kp1, km1
  # Receptor-aggregation
  2 EGFR(L!+,CR1) + EGFR(L!+,CR1) <-> EGFR(L!+,CR1!1).EGFR(L!+,CR1!1) kp2,km2
  # Transphosphorylation of EGFR by RTK
  3 EGFR(CR1!+,Y1068~U) -> EGFR(CR1!+,Y1068~P)  kp3
  # Dephosphorylation
  4 EGFR(Y1068~P) -> EGFR(Y1068~U) km3
  # Grb2 binding to pY1068
  5 EGFR(Y1068~P) + Grb2(SH2) <-> EGFR(Y1068~P!1).Grb2(SH2!1) kp4,km4
  # Grb2 binding to Sos1
  6 Grb2(SH3) + Sos1(PxxP) <-> Grb2(SH3!1).Sos1(PxxP!1) kp5,km5
  # Receptor dimer internalization/degradation
  7 EGF(R!1).EGF(R!2).EGFR(L!1,CR1!3).EGFR(L!2,CR1!3) -> Trash() deg\
  DeleteMolecules
end reaction rules

#actions
generate_network({overwrite=>1});
# Equilibration
simulate_ode({suffix=>equil,t_end=>100000,n_steps=>10,sparse=>1,\
    steady_state=>1});
setConcentration("EGF(R)","EGF_init");
saveConcentrations(); # Saves concentrations for future reset
# Kinetics
writeSBML({});
simulate_ode({t_end=>120,n_steps=>120});
resetConcentrations(); # reverts to saved Concentrations
simulate_ssa({suffix=>ssa,t_end=>120,n_steps=>120});
```

The five block types are "`parameters`," "`molecule types`," "`seed species`," "`reaction rules`," and "`observables`." The blocks may appear in any order. Actions to be performed on the model are controlled using commands that follow the model specification. All text following a "#" character on a line is treated as a comment, and comments may appear anywhere in an input file. Parsing of the input is line-based, and a continuation character, "\", is required to extend a statement over multiple lines. There is no limit on line length. Any BioNetGen input line may begin with an integer index followed by space, which is ignored during input processing but may be useful for reference purposes. For example, .net files produced by BioNetGen automatically index elements of each input block.

The following is a list of the general steps involved in constructing a BioNetGen model with the relevant section of the BNGL input file shown in parenthesis:

1. `(parameters)` Define the parameters that govern the dynamics of the system (rate constants, the values for initial concentrations of species in the biological system) (*see* **Subheading 3.1**).
2. `(molecule types)` Define molecules, including components and allowed component states (*see* **Subheading 3.2**).
3. `(seed species)` Define the initial state of system (initial species and their concentrations) (*see* **Subheading 3.3**).
4. `(observables)` Define model outputs, which are functions of concentrations of species having particular attributes (*see* **Subheading 3.4**).
5. `(reaction rules)` Define rules that describe how molecules interact (*see* **Subheading 3.5**).
6. `(actions)` Pick method(s) for generating and simulating the network (*see* **Subheading 3.6**).

Steps 1–5 may be done in any order and the entire protocol is likely to undergo multiple iterations during the process of model development and refinement. **Subheadings 3.1–3.6** describe the sections of the BNGL input file with specific reference to the model presented in **Listing 1**. **Subheading 3.7** then presents two additional models that illustrate the use of more advanced language features.

3.1. Parameters

Model parameters, such as rate constants, values for initial concentrations of chemical species, compartment volumes, and physical constants used in unit conversions can be defined in the `parameters` block (*see* **Note 3**). Both numerical and formula-based parameter assignments are illustrated in the `parameters` block of **Listing 1**, which illustrates how formulas may be used to clarify unit conversions and to define a global parameter that

controls the system size (*see* **Note 4**). Parameters have no explicitly defined units, but must be specified in consistent units, as assumed by BioNetGen. We recommend that concentrations be expressed in units of copy number per cell and bimolecular rate constants be expressed on a per molecule per cell basis, as in **Listing 1**. This choice, which assumes that the reaction compartment is a single cell and its surrounding volume, allows one to direct BioNetGen to switch from a deterministic simulation to a stochastic simulation without changing parameter units.

3.2. Molecule Types

Molecules in a BioNetGen model are structured objects composed of components that can bind to each other, both within a molecule and between molecules. Components typically represent physical parts of proteins, such as the SH2 and SH3 domains of the adapter protein Grb2, or the PxxP motif of the guanine nucleotide exchange factor Sos1 that serves as a binding site for SH3 domains. Components may also be associated with a list of state labels, which are intended to represent states or properties of the component. Examples of component states that can be modeled using state labels are conformation (e.g., open or closed), phosphorylation status, and location. There is no limit on the number of components that a molecule may have or on the number of possible state labels that may be associated with a component (*see* **Note 5**).

BioNetGen allows users to *explicitly* enforce typing of molecules using the `molecule types` block, which is optional but recommended. The `molecule types` block defines the allowed molecule names, the components of each molecule type (if any), and the allowed states of each of these components (if any). Each molecule type declaration begins with the name of a molecule (*see* **Note 6**) followed by an optional list of components in parentheses (*see* **Note 7**). The tilde character ("~") precedes each allowed state value. In the input file of **Listing 1**, five molecule types are declared. These molecules have 1, 3, 2, 1, and 0 components, respectively. The component named Y1068 represents a tyrosine residue in EGFR that can be in either an unphosphorylated (U) or phosphorylated (P) state. For a molecule to be able to bind another molecule, at least one component must be defined. A molecule without components cannot bind or change states, but can be created or destroyed. Such a molecule essentially corresponds to a named chemical species in a conventional model (*see* **Subheading 3.5.6**). A component that appears in a molecule type declaration without a state label may be used only for binding and may not take on a state label in subsequent occurrences of the same molecule. In contrast, the potential binding partners of a component are not delimited in a molecule type declaration.

The namespaces for components of different molecules are separated, so it is permissible for components of different molecules

to have the same name. If two components of the same molecule have the same name, however, they are treated as separate instances of an identical type of object. For example, the two Fab arms of an IgG antibody have identical antigen-binding sites, which could be modeled as `IgG(Fab,Fab)`.

3.3. Seed Species

The `seed species` block defines the initial chemical species to which rules are applied. This block may also be used to define the initial levels of populated species and identify species with fixed concentrations. Before discussing the details of the `seed species` block, we need to briefly explain how chemical species are represented in BNGL.

Chemical species are individual molecules or sets of molecules connected by bonds between components, in which each component that has allowed state values has a defined state. For example, a cytosolic complex of Grb2 and Sos1 in the model of **Listing 1** would be represented as `Grb2(SH2,SH3!1).Sos1(PxxP!1)`, where the "." character is used to separate molecules that are members of the same chemical species and the "!" character is a prefix for a bond name (any valid name is allowed, but we recommend using an integer, which makes BNGL expressions more readable). A shared name between two components indicates that the components are bonded. A complex of Grb2 and Sos1 that is associated with EGFR would be represented as `EGFR(L,CR1,Y1068~P!2). Grb2(SH2!2,SH3!1).Sos1(PxxP!1)`, where the bond with the name "2" in this expression indicates that the SH2 domain of Grb2 is connected to the phosphorylated residue Y1068 in EGFR (i.e., connected to component `Y1068` of the molecule `EGFR`, which is in the `P` state). Note that in a BNGL expression for a chemical species all components of each molecule are listed and each component that is allowed to have a state has one defined state chosen from among the set of possible states for that component. Wild card characters, which represent nonunique states and bonds, are not allowed in BNGL chemical species expressions. These wild card characters are discussed below in **Subheading 3.4**.

Finally, we note that the presence or absence of the `molecule types` block affects the way that molecules appearing in the `seed species` block are type checked (*see* **Note 8**).

Specifying the initial population level of a seed species is accomplished in the same way that a parameter value is assigned using either a numerical value or a formula, as can be seen in **Listing 1** (*see* **Note 9**). A species listed in the `seed species` block may also be designated as having fixed concentration (*see* **Note 10**).

Representation of molecular complexes in BNGL has been presented in this section to introduce the syntax of bonds, but, generally speaking, it is not necessary to define seed species that are complexes of molecules because they can be generated through a process of

equilibration (*see* **Subheading 3.6**), provided that there are rules that generate these complexes. If a complex species is defined and no reaction rule is specified that causes dissociation of the complex, the complex will be indivisible. A multimeric protein composed of several polypeptide chains could be specified in this way.

3.4. Observables

The `observables` block is used to specify model outputs, which are functions of the population levels of multiple chemical species that share a set of properties. For example, if one could measure the tyrosine phosphorylation level of a particular protein, then one might be interested in determining the total amount of all chemical species containing the phosphorylated form of this protein. We call a function for calculating such a quantity an "observable." Observables are computed over a set of chemical species that match a search pattern or set of search patterns specified in BNGL (*see* **Fig. 1b**). Each observable is defined by a line in the `observables` block consisting of an (optional) index, one of two keywords that defines the type of observable (Molecules or Species), a name for the observable, and a comma-separated set of search patterns (*see* **Listing 1** and **Note 11**). Before we discuss the two types of observables and how they are computed, we will describe the basic syntax and semantics of patterns in BNGL, which are common to observables and reaction rules.

Patterns are used to identify a set of species that share a set of features, and their behavior is illustrated in **Fig. 1b**. Pattern specification includes one or more molecules with optional specification of connectivity among these molecules, optional specification of states of their components, and optional specification of how these molecules are connected to the rest of the species they belong to. Patterns are analogous to the regular expressions used in computer programming. A match between a chemical species and a pattern means that there exists a mapping (injection) from the elements of the pattern to a subset of the elements of the species. Roughly speaking, a species matched by a pattern includes this pattern as a part. Note that there may be multiple mappings of a pattern into a single species and that BioNetGen considers each mapping to be a separate match. The formal definition of a match in the graph formalism upon which BNGL is based was given by Blinov et al. *(26)*. Patterns are similar to species in that they are composed of one or more molecules and may contain components, component state labels, and edges. Unlike in species, however, the molecules in patterns do not have to be fully specified and the molecules do not have to be connected to each other by bonds specified in the pattern. The absence of components or states in a pattern excludes consideration of the missing elements from the matching process, as illustrated in **Fig. 1b**. In the model of **Listing 1** observable 1 is specified using the pattern `EGFR()`, which matches any species containing a molecule of EGFR, regardless of the state or binding status of any of its components.

When a component is specified in a pattern, both the absence and presence of a bond name affects matching. The specification of a component without an associated bond requires that the component is unbound in the corresponding match. For example, observable 2 in **Listing 1** uses the pattern, `EGF(R)`, which selects only species in which the R component of EGF is unbound. The specification of a component with an associated bond is used to select bound components. If a complete bond is specified, as in observable 5, which selects complexes of Grb2 and Sos1, then the component must be bound in the manner indicated by the pattern (*see* **Note 12**). An incomplete bond may also be specified using "!+", where the wild card "+" indicates that the identity of the binding partner of a component is irrelevant for purposes of matching. For example, observable 3 in **Listing 1** uses the pattern, `EGFR(L!+)`, which selects species in which the L component of EGFR is bound, regardless of the binding partner. A second wild card, "?", may be used to indicate that a match may occur regardless of whether a bond is present or absent (*see* **Note 13**), and is sometimes required for the correct specification of observables. For example, the two patterns `EGFR(Y1068~P)` and `EGFR(Y1068~P!?)` are not equivalent. The first pattern selects only EGFR molecules in which the Y1068 component is phosphorylated and unbound, whereas the second pattern selects all EGFR molecules in which the Y1068 component is phosphorylated. (The second pattern is more relevant for comparing model predictions against the results of Western blotting with anti-pY antibodies.) Examples of patterns from the `observables` block of **Listing 1** and their corresponding matches in the implied model are listed in **Note 14**.

We are now ready to discuss the two types of observables. An observable of the Molecules type is a weighted sum of the population levels of the chemical species matching the pattern(s) in the observable. Each population level is multiplied by the number of times that the species is matched by the pattern(s). An observable of the Species type is simply an unweighted sum of the population levels of the matching chemical species (*see* **Notes 15** and **16**). A Molecules type of observable is useful for counting the number of copies of a particular set of patterns in a system, e.g., the number of copies of receptors in receptor dimers. A Species type of observable is useful for counting the populations of chemical species in a system containing a particular pattern (or set of patterns), e.g., the number of receptor dimers, as specified by observable 3 in **Listing 1**. Changing the type from Species to Molecules for this observable would specify a function that gives the number of copies of receptors in receptor dimers. The values of observables computed by one of the simulation commands described below are written to a .gdat file (*see* **Note 17**).

3.5. Reaction Rules

The `reaction rules` block of a BioNetGen input file is used to specify rules, which describe the allowed ways in which species can be transformed and typically represent molecular interactions and the consequences of these interactions. Each rule is similar to standard chemical reaction notation in that it has four basic elements: reactant patterns, an arrow, product patterns, and a rate law specification (*see* **Note 18**). Patterns in rules have the same syntax and semantics as introduced above in our discussion of the `observables` block. Reactant patterns are used to select sets of reactant species to which the transformation implied by the rule will be applied. The arrow indicates whether the rule is applicable in forward direction only ("–>") or in both the forward and reverse directions ("<–>"). The product patterns define how the selected species are transformed by the rule and act as the reactant patterns when the rule is applied in reverse. Rules may transform a selected set of reactant species by adding or deleting molecules or bonds and by changing component state labels. Rules may not add or delete components of molecules (*see* **Note 19**). The default rate law for reactions produced by rules is an elementary rate law, in which the rate is given by the product of a multiplicity factor (usually an integer or ½) generated automatically by BioNetGen (*see* **Subheading 3.5.3**), the specified rate constant (which may be a numerical value or a formula), and the population levels of the reactants. This type of rate law is specified simply by appending a comma-separated numerical value or formula at the end of the line defining a rule, as illustrated in **Listing 1**. Nonelementary rate laws, such as Michaelis–Menten rate laws, may also be specified (*see* **Note 20**). For a rule that defines reverse reactions, a second numerical value or formula follows the first after a comma. Rules 1, 2, 5, and 6 in **Listing 1** provide examples of how the parameters of two elementary rate laws are defined on the same input line. It should be noted that the parameter of a default rate law is taken to be a single-site rate constant (*see* **Note 21**). Additional commands that modify the behavior of rules may appear after the rate law specification (*see* **Subheading 3.5.7**).

Consider the `egfr_simple.bngl` file illustrated in **Listing 1**. Each reaction rule is defined on one line of the input file. (Recall that long input lines can be continued using the "\" character.) The first six rules represent classes of reactions mediated by particular molecular interactions (e.g., rule 1 specifies a class of ligand-receptor binding reactions in which the R domain of the ligand associates with the L domain of the receptor), and the last rule represents a class of irreversible degradation reactions, which removes receptor dimers from the system while retaining cytosolic molecules bound to the receptor complex. Rules 3 and 4 also define classes of irreversible reactions, whereas the remaining rules define classes of reversible reactions. The molecularity of

a reaction, *M*, is the number of species participating in the reaction. The molecularity of all reactions generated by a given rule is fixed and is equal to the number of reactant patterns, which are separated by "+" characters. The value of *M* for rules 1–*7* in **Listing 1** is 2, 2, 1, 1, 2, 2, and 1, respectively. The "+" character is used on the right side of a rule to define the number of products produced by a reaction and the molecularity of reverse reactions (if the rule is reversible). In **Listing 1**, rules 1–6 each have one product, and the reverse reactions have 2, 2, 1, 1, 2, and 2 product(s), respectively. Reactions defined by rule *7* have a variable number of products because of the `DeleteMolecules` keyword, which is discussed later in this section.

We will now discuss the five basic transformations that can be carried out by a BioNetGen rule. These transformations are (1) add a bond, (2) delete a bond, (3) change a component state label, (4) delete a molecule, or (5) add a molecule. In each case, there is a direct correspondence between a transformation of a set of graphs and a biochemical transformation of the molecules represented by the graphs *(26)*. For example, adding a bond between the interacting components of two binding partners corresponds to connecting two vertices in the graphs representing these binding partners. In the following subsections, we will discuss each of these types of transformations and present examples. A transformation is specified implicitly by the difference between the product and reactant patterns in a rule. BioNetGen automatically determines a mapping from reactant molecules and their components to product molecules and their components, and from this mapping determines the set of transformations implied by a rule. Although we will note exceptions, we recommend in general that each rule apply only a single transformation. A user may manually override automatic mapping through the use of molecule and component labels, as discussed in **Subheading 3.7.1** (*see* **Note 22**) *(28, 38)*. Such labels have been used to create a database of carbon atom fates in metabolic reactions *(38)*.

3.5.1. Add a Bond

A rule may add bond labels (e.g., "`!1`") to specific components of reactant species selected by the reactant pattern(s) in the rule, which results in the formation of a new bond. Including a bond in a product pattern that is absent in the reactant pattern(s) specifies this action. The simplest example of such a transformation is provided by the rule "`A(a)+B(b)->A(a!1).B(b!1) k_bi,`" which specifies the association of molecules A and B through the formation of a bond between components a in molecule A and b in molecule B. Note that the "+" character constrains the molecularity to 2, which means that a and b must belong to separate species, precluding binding of A to B when these molecules are part of the same complex. To specify intracomplex binding of a and b, we could specify the rule as "`A(a).B(b)<->A(a!1).B(b!1)`

k_uni", where the "." character in the reactant pattern indicates that the molecules A and B are part of the same complex. Note that these two rules have bimolecular and unimolecular rate laws, respectively, because they have different molecularities, and thus the units of k_bi and k_uni necessarily differ. As noted earlier, it is the modeler's responsibility to specify values of model parameters using consistent units.

Let us consider rule 1 in **Listing 1**, which provides an example of a reaction rule for the reversible binding of a ligand to a receptor. We first consider application of the rule in the forward direction (application of the rule in reverse will be considered in **Subheading 3.5.2**). The reactant pattern EGF(R) selects ligand (EGF) molecules that have an unbound R component. Since EGF molecules in this model have only one component, the only species that is selected by this pattern is **EGF(R)** (Here, we adopt the convention that the image of a pattern in a matching species is shown in **bold**). The pattern EGFR(L,CR1) selects EGFR molecules with unbound L and CR1 components, regardless of the binding or phosphorylation status of the Y1068 component of EGFR. For example, the pattern would select all of the following possible species: **EGFR(L,CR1,**Y1068~U), **EGFR(L,CR1,**Y1068~P), and **EGFR(L,CR1**,Y1068~P!1). Grb2(SH2!1,SH3). By specifying the component CR1 in the pattern and indicating that this component is free (by the absence of a bond specification), we are requiring that the CR1 component be unbound. Because receptors must associate via the CR1 domain to form dimers, as specified by rule 2, this means that ligand can bind receptor monomers but not dimers through rule 1. Rule 1 can be made independent of the state of CR1 by simply omitting it from the pattern for EGFR. In other words, by specifying EGFR(L) instead of EGFR(L,CR1), ligand is allowed to associate with (and dissociate from) both monomeric and dimeric receptors. The general principle is that a reaction rule should only include molecules, components, state labels, and bond specifications that are either modified by a transformation or that affect the transformation. We call the component(s) directly modified by a transformation a *reaction center* and the rest of the information included in a rule the *reaction context*. For clarity, we will underline the reaction centers in the rules (*see* **Listing 1**). The process of rule application is illustrated in **Fig. 1** and further examples are listed in **Note 23**.

Let us now consider rule 2 of **Listing 1**, which specifies the reversible dimerization of ligand-bound EGFR and illustrates the use of bond wild cards in the reactant specification. The "!+" string following the L component of each EGFR means that the L component must be bound (albeit in an unspecified way) for the pattern to match and thus for the reaction to take place.

Another important feature of this rule is that it is symmetric with respect to interchange of the two reactant patterns, which is detected automatically by BioNetGen, which then ensures that generated reactions are assigned rate laws with correct multiplicity. Reaction multiplicity, which is a multiplicative factor in a rate law, is discussed in more detail below in **Subheading 3.5.3**. For many users, it is sufficient to note that BioNetGen automatically detects symmetries in rules and generates reactions with correct multiplicities.

3.5.2. Delete a Bond

Rules specify bond deletion when a bond that appears in the reactant patterns has no corresponding bond on the product side (*see* **Note 24**). Frequently, bond deletion rules are specified simply by making a bond addition rule reversible, as in the extension of the elementary bond addition rule above to "`A(a)+B(b) <-> A(a!1).B(b!1) k_a,k_d`". Bond dissociation step can also be specified using a unidirectional rule, as in "`A(a!1).B(b!1) -> A(a)+B(b) k_d`". The reversible rule syntax is provided solely as a matter of convenience; the functional behavior of the rules is identical whether an association/dissociation pair is specified as a single reversible rule or as two irreversible rules with the reactant and product patterns interchanged (*see* **Note 25**). Note that the molecularity of the products in the dissociation rule (2 in this case) has a restrictive effect analogous to that of the specification of molecularity in the association rule. When the rule is applied to a species selected by the reactant pattern, a reaction is generated only if removal of the specified bond eliminates all possible paths along bonds between A and B, i.e., if bond removal produces two separate fragments. Specifying bond dissociation that does not result in breakup of the complex requires a rule of the form "`A(a!1).B(b!1) -> A(a).B(b) k_d`". An example illustrating the different action of these two rules is provided in **Note 26**.

As an example of a bond deletion rule that has additional reaction context, let us consider the reverse of the dimerization rule discussed in **Subheading 3.5.1**,

```
EGFR(L!+,CR1!1).EGFR(L!+,CR1!1)->EGFR(L!+,CR1)+\
                                 EGFR(L!+,CR1) km2,
```

which breaks the bond between the CR1 components of two receptors in a complex. The contextual requirement that an L component of each EGFR also be bound is specified using the bond wild card "`L!+`". The molecularity of the products in the rule means that the rule will only be applied if breaking the bond results in dissociation of an aggregate. It is important to note here that the bond wild card "`!+`"can only be used to specify context; it is not permitted to break a bond that is only partially specified because such a rule would leave the molecularity unspecified.

3.5.3. Change a Component State Label

Rules specify a change in the state label of a component whenever the state label of a component changes in going from its appearance in the reactants to its corresponding occurrence in the products. State label changes may be used to represent covalent modification, a change in conformation, translocation between two compartments, or any other property of a molecule that might influence its subsequent reactivity. The simplest possible example of a rule specifying a state label change is rule 4 of **Listing 1**,

```
EGFR(Y1068~P)->EGFR(Y1068~U) km3
```

which encodes the dephosphorylation of a receptor tyrosine, through a change in the state label for Y1068 from "P", representing the phosphorylated state, to "U", representing the unphosphorylated state (*see* **Note 27**). It should be noted that just as for bond addition and deletion reactions, the rate constant should be specified as if only one instance of the reaction implied in the rule is possible for any given set of reactant species (*see* **Note 21**). BioNetGen will generate a distinct reaction for each distinct occurrence of the reactant pattern in a species. For example, consider the application of rule 4 to the following species in the EGFR network:

```
EGF(R!1).EGF(R!2).EGFR(CR1!3,L!1,Y1068~P)\
.EGFR(CR1!3,L!2,Y1068~P)
```

The two occurrences of the reactant pattern are shown in **bold**. During the process of network generation, this species is automatically assigned the index 11, which is used to reference species in the `reactions` and `groups` blocks of the resulting .net file. Because this species is symmetric, application of the rule generates two instances of the dephosphorylation reaction 11 → 8, and species 8 is

```
EGF(R!1).EGF(R!2).EGFR(CR1!3,L!1,Y1068~U)\
.EGFR(CR1!3,L!2,Y1068~P)
```

In this case, application of rule 4 to the first Y1068 appearing in species 11 generates the same species as application of the rule to the second instance (*see* **Note 28**). Upon generation of a reaction, BioNetGen checks to determine whether the reaction is identical to one that has already been generated. If so, the multiplicity of the reaction is incremented by one (*see* **Note 29**). So application of rule 4 to species 11 produces the reaction `11→ 8 2*km3`, where `2*km3` following the reaction refers to the constant portion of the elementary rate law that is used to compute the rate of the reaction. The multiplicity of the reaction is 2, and the rate is given by `2*km3*X11`, where X11 is the population level of species 11.

As in other reaction rules, additional contextual information can be supplied to restrict application of a rule. An example of a rule that uses contextual information in this way is rule 3 of **Listing 1**, which specifies phosphorylation of Y1068 within a receptor aggregate:

```
EGFR(CR1!+,Y1068~U)→ EGFR(CR1!+,Y1068~P) kp3
```

In this rule, the wild card operator "+" is used to specify that the phosphorylation reaction occurs only for a receptor that is part of a receptor dimer. Because in this model the CR1 domain can only bind to another CR1 domain, requiring CR1 to be bound, as specified here, is equivalent to requiring that another EGFR be present in the aggregate (*see* **Note 30**). Thus, the rule above models trans (auto)phosphorylation of Y1068 catalyzed by the protein tyrosine kinase domain in a neighboring copy of EGFR.

3.5.4. Add a Molecule

In addition to the operations described in the previous sections, rules may also specify the creation of new molecules as products, which could be used to model, for example, translational processes or transport across the cell membrane. As a simple example of how to introduce a source for a protein A, consider the rule

```
I()-> I()+A(a,Y~U) ksynth
```

where `I()` is a structureless molecule. The appearance of I on both the reactant and product sides of the rule means that its concentration will not change as a result of the reaction occurring. If the species "`I()`" is set to have a concentration of 1 in the `seed species` block and its concentration is not affected by any other rules, the rate constant `ksynth` will be have units of concentration/time and will define the synthesis rate of the species `A(a,Y~U)`. Note that BioNetGen does not allow the number of reactants or products in a reaction to be zero, which is why the molecule I must be included in this rule. Molecule addition is specified any time that a molecule appearing on the product side of a rule has no corresponding molecule on the reactant side. Appearance of a new molecule in the products generates an error unless the molecule is fully specified, i.e., all components of the molecule are listed and those components requiring a state label have a valid specified state label, and connected to the remainder of the pattern in which it appears. New molecules can also be combined with reactant molecules, as in the rule

```
B(b)-> B(b!1).A(a!1,Y~U) ksynth
```

which creates a new molecule of A bound to a B molecule.

3.5.5. Delete a Molecule

Rules may also specify degradation of specified molecules or of entire species matching a particular reactant pattern by omitting reactant molecules in the product patterns. Because degradation rules may specify deletion of individual molecules or entire species, the semantics of degradation rules are somewhat more complicated than those of other rules considered so far. Let us first consider the simplest form of a degradation rule

```
A() -> Trash() kdeg
```

which specifies degradation of any species in which the molecule A appears. Degradation of a species is specified whenever all of the reactant molecules used to select the species are omitted from the products. This rule also specifies the synthesis of a Trash molecule, which is necessary because BioNetGen require that at least one product molecule be specified. Note that the species `Trash()` acts as a counter for the number of A-containing species that have been degraded (*see* **Note 31**). If multiple molecules of A can appear within a single species, degradation reactions involving these species would have multiplicity equal to the number of occurrences of A in the degraded species. In other words, a species containing n copies of A will be degraded n times faster than a species containing only a single copy of A. If this behavior is not the desired, then the multiplicity can be held to one by specifying the `MatchOnce` attribute for the reactant pattern, as in

```
{MatchOnce}A() -> Trash() kdeg
```

As of this writing `MatchOnce` is the only recognized pattern attribute.

Rules can also specify the degradation of a set of molecules within a complex, which can be accomplished in one of two ways. First, one can specify the degradation of a molecule or molecules within a reactant complex by transferring to the products at least one of the molecules used to select the complex on the reactant side. The simplest example is the rule

```
A().B() -> B() kdeg
```

which specifies the deletion of the matching A molecule in the complex. When the rule is applied, the A molecule and all of its bonds will be deleted. If this action leaves behind only a single connected fragment containing the matched B molecule, a reaction will be generated. If, however, deletion of A leaves behind multiple fragments, no reaction will be generated. The keyword `DeleteMolecules` can be added to the rule following the rate law to bypass this constraint, as in

```
A().B()-> B()kdeg DeleteMolecules
```

which, when applied to the complex `C(c!1).A(a1!1,a2!2).B(b!2)`, would generate the reaction

```
C(c!1).A(a1!1,a2!2).B(b!2)->C(c)+B(b)kdeg
```

The deletion of the A molecule from the C-A-B chain produces a C fragment and a B fragment. The `DeleteMolecules` keyword can also be used when no molecules from the reactant pattern remain in the products. Thus, the species-deleting rule from the previous paragraph can be transformed into a molecule-deleting rule

```
A()->Trash() kdeg DeleteMolecules
```

which has the same action on the C-A-B complex as the rule above.

Rule 7 of **Listing 1** provides an example of how such a rule might be used to model endosomal degradation of signaling complexes in which some components of the complex are recycled. The rule specifies that the EGFR dimer and both associated EGF molecules are degraded, but the `DeleteMolecules` keyword means that additional molecules associated with the complex will be retained as products in any generated reactions. Thus, any Grb2 molecules that associate with such a dimer and any Sos1 molecules that bind to dimer-associated Grb2 molecules are (effectively) returned to the cytoplasm when the receptor complex is degraded.

3.5.6. Encoding Conventional Reactions

Addition and deletion actions may be combined within single rules to construct rules that describe conventional mass action kinetics involving structureless species. A typical rule of this type would be

```
A+B -> C kAB
```

which encodes the deletion of A and B and the addition of C. This rule will be valid only if the molecule C is defined to have no components, and it will have the intended meaning only if A and B are also structureless. Any standard reaction scheme can thus be trivially encoded in BioNetGen, although the power of the rule-based approach is lost. Structureless species may be useful as sources and sinks, and may also be used to represent small molecules or atoms. Note that `A` and `A()` are equivalent representations for a molecule or species A, in that neither representation specifies the substructure of A.

3.5.7. Commands for Modifying Rule Application

As described in **Subheading 3.5.5**, BioNetGen includes several commands that modify the application of rules. These commands have been introduced to address the need for specific behaviors

that are difficult or impossible to specify using the semantics of patterns and transformation rules alone. In this section, we cover the `include/exclude` commands that provide a basic logic for extending the selection capabilities provided by patterns. In the future, we anticipate the development of a "pattern logic" that will provide these capabilities in a more general way.

The basic functionality of the `include_reactants` and `include_products` commands is to add criteria for the selection of reactant species to be transformed by a rule or the acceptance of products species generated by a rule. In other words, these commands provide an AND operator for pattern matching. The basic syntax of the include commands is illustrated by the rule

```
A(a)+B(b)->A(a!1).B(b!1) kabinclude_reactants\
(2,R1,R2)
```

which specifies that a bond will be created between a reactant species containing a free component a of a molecule A and a second reactant species containing a free component b of a molecule B only if the second reactant species also includes a molecule of *either* R1 or R2. The first argument of an `include` command is always a number corresponding to the index of a reactant or product pattern in the rule (1 for the first reactant/product, 2 for the second, etc.), and the remaining arguments are BNGL patterns, at least one of which must generate a match for the species to be selected. In logical terms, the effective pattern for the second reactant in this rule becomes "`B(b) AND (R1 OR R2)`". Any valid BioNetGen pattern may be used as an argument to an `include_reactants` or `include_products` command. Multiple include commands applying to the same reactant or product pattern can be specified to create additional selection criteria for a species, and thus function as additional AND operators. To generate similar behavior without the include command, two rules would have to be specified:

```
A(a)+B(b).R1A->(a!1).B(b!1) kab
A(a)+B(b).R2A->(a!1).B(b!1) kab
```

where the "." operator is used to test for the presence of an additional molecule in the second reactant complex. It is worth noting that the rule using the `include_reactants` command behaves slightly differently in this case than the pair of rules, because the latter may each generate multiple matches to the same reactant species if multiple molecules of either R1 or R2 are present. For instance, the pattern "`B(b).R1`" generates two matches to the species "`B(b.r!1).R1(r!1,d!2).R1(r,d!2)`" because R1 in the pattern can be mapped onto either of the two R1's in the complex. It is easy to specify two rules that have the same behavior as the

one rule by extending the pattern "`B(b).R1`" to "`B(b.r!1).R1(r!1)`". Unfortunately, as illustrated in this example, subtle differences in the way that rules are specified can have dramatically different effects, which are sometimes difficult to anticipate. This problem will be alleviated in the future by extending BioNetGen to allow a user to differentiate between the reaction center (the part of a pattern affected by a transformation) and reaction context (the part of a pattern necessary for a transformation to occur) in rules.

The "`exclude_reactants(index,pattern1,pattern2,...)`" and "`exclude_products`" commands have the same syntax as the `include` commands but apply the logic "pattern_index AND ((NOT pattern1) OR (NOT pattern2) …)", where pattern_index is the pattern used to specify the reactant or product with the specified index. Equivalent functionality can be obtained by the use of patterns alone, but in complex cases several patterns may be required to accomplish the same effect. It should be noted that when they appear in reversible reactions, include_reactants and exclude_reactants are automatically transformed into `include_products` and `exclude_products`, respectively, when the rule is applied in the reverse direction. Appearances of `include_products` and `exclude_products` commands are also similarly transformed.

3.6. Actions

BioNetGen is capable of performing two basic types of actions with a model specification in an input file: generate a chemical reaction network implied by the model specification and simulate the network (e.g., solve an initial value problem for the system of coupled ODEs that provides a deterministic description of the reaction kinetics in the well-mixed limit). These actions are controlled using commands that follow the model specification blocks we have discussed in the previous section (*see* **Listing 1**). Other commands export BioNetGen-generated networks in various formats. All of the available commands and the parameters that control them are summarized in **Table 1**, which also summarizes the general syntax.

3.6.1. Generating a Network

The commands shown in **Listing 1** illustrate the range of actions that can be performed on a BioNetGen model. The `generate_network` command directs BioNetGen to generate a network of species and reactions through iterative application of the rules starting from the set of seed species. At each step in this iterative process, rules are applied to the existing set of chemical species to generate new reactions. Following rule application, the species appearing as products in the new reactions are checked to determine whether they correspond to existing species in the network *(26)* (*see* **Note 32**). If no new species are found, network generation terminates.

Restrictions on rule application may be useful when rules sets would otherwise produce very large or unbounded networks (*see*

Table 1
Syntax and parameters for BioNetGen actions

Action/parameter[a]	Type[b]	Description	Default
generate_network		**Generate species and reactions through iterative application of rules to seed species**	
max_agg	int	Maximum number of molecules in one species	1e99
max_iter	int	Maximum number of iterations of rule application	100
max_stoich	hash	Maximum number of molecules of specified type in one species	-
overwrite	0/1	Overwrite existing .net file	0 (off)
print_iter	0/1	Print .net file after each iteration	0
prefix[c]	string	Set basename[c] of .net file to *string*	basename of .bngl file
suffix[c]	string	Append *_string* to basename of .net file	-
simulate_ode/simulate_ssa		**Simulate current model/network**	
t_end	float	End time for simulation	required
t_start	float	Start time for simulation	0
n_steps	int	Number of times after t=0 at which to report concentrations/observables	1
sample_times	array	Times at which to report concentrations/observables (supercedes t_end, n_steps)	-
netfile	string	Name of .net file used for simulation	-
atol[d]	float	Absolute error tolerance for ODE's	1e-8
rtol[d]	float	Relative error tolerance for ODE's	1e-8
steady_state[d]	0/1	Perform steady-state check on species concentrations	0
sparse[d]	0/1	Use sparse Jacobian/iterative solver (GMRES) in CVODE	0
readFile		**Read a .bngl or a .net file**	
file	string	Name of file to read	required
writeNET/writeSBML/ writeMfile		**Write current model/network in specified format**	
setConcentration(species,value)		**Set concentration of species to value**	
setParameter(parameter,value)		**Set parameter to value**	

(continued)

Table 1
(Continued)

Action/parameter[a]	Type[b]	Description	Default
saveConcentrations()		Store current species concentrations	
resetConcentratons()		Restore species concentrations to value at point of last save-Concentrations command	

[a]General syntax is *action*({*scal val*,*array* [*x1*,*x2*,…],*hash* ⇒{*key1*⇒*val1*,*key2*⇒*val2*,…},…}).
[b]Scalar types are int, 0/1 (a boolean), string, and float. Multivalued parameters may be either arrays or hashes.
[c]The prefix and suffix parameters can be used with any command that writes output to a file.
[d]These parameters only apply to simulate_ode.
[e]*See* **Note 35**.

Note 33). These restrictions can be imposed using optional arguments to the `generate_network` command, which are shown in **Table 1**. The three basic restrictions that can be specified are an upper limit on the number of iterations of rule application (`max_iter`), an upper limit on the number molecules in an aggregate (`max_agg`), and an upper limit on the number of molecules of a particular type in an aggregate (`max_stoich`). An example of a command specifying all three restrictions in the order given above is

```
generate_network ({max_iter => 15,max_agg => 10,
                   max_stoich=>{L => 5,R > 5});
```

This command limits the number of iterations to 15, the maximum size of an aggregate to 10 molecules, and the maximum number of L or R molecules in an aggregate to be 5. An example illustrating the use of such restrictions is given in **Subheading 3.7.2**.

When network generation terminates, whether through convergence or when a stopping criterion is satisfied, the resulting network is written to a file with the .net extension (*see* **Note 34**). By default the basename of this file is determined from the basename of the input .bngl file. For example, the `generate_network` command in the file `egfr_simple.bngl` creates the file `egfr_simple.net` by appending the .net extension to the basename `egfr_simple.` The options `prefix` and `suffix`, which are taken by all commands that write output to a file, can be used to modify the basename of all files generated by the command (*see* **Note 35**). By default, `generate_network` will terminate with an error if the .net file it would produce exists prior to network generation. This behavior can be overridden by setting option `overwrite =>1`, as shown in **Listing 1**. This option can be useful during the debugging phase of model development.

3.6.2. Simulating a Network

Once a network has been generated, a simulation can be specified using the `simulate_ode` or `simulate_ssa` commands. The simulation specified in the example in **Listing 1** consists of three phases, which we now summarize and will be described in detail below. The first phase is equilibration, in which reactions that can occur prior to the introduction of the EGF ligand are allowed to reach steady state. Time courses produced by the first `simulate_ode` command, which terminates when the species concentrations pass a numerical check for convergence, are written to the files `egfr_simple_equil.gdat` and `egfr_simple_equil.cdat` (assuming the input file is named `egfr_simple.bngl`). Before the second phase of simulation, ligand is introduced (using `setConcentration`), the concentrations at the end of equilibration are saved (using `saveConcentrations`), and the network is written to an SBML file (using `writeSBML`). The second `simulate_ode` command then initiates a simulation of the dynamics following introduction of EGF ligand into the system. The results are written to the files `egfr_simple.gdat` and `egfr_simple.cdat`. The third phase is then preceded by a `resetConcentrations` command, which restores the concentrations to the initial values used in the second phase, i.e., following equilibration and introduction of EGF. The `simulate_ssa` command then initiates the third and final phase of simulation, a kinetic Monte Carlo simulation using the Gillespie algorithm, and results are written to the files `egfr_simple_ssa.gdat` and `egfr_simple_ssa.cdat`.

In the equilibration phase the population level of the ligand (`EGF(R)`) is zero, as specified in the `seed species` block of **Listing 1**. Network generation is unaffected by the population levels of the seed species, but in the absence of ligand the only reactions with nonzero flux are the binding and unbinding reactions of Grb2 and Sos1 in the cytosol, which are defined by rule 6. The purpose of the equilibration phase is then to allow the concentrations of free Grb2, free Sos1, and the cytosolic Grb2-Sos1 complex to reach steady-state levels, which we would expect to find in the resting state of the cell.

The first `simulate_ode` command propagates the simulation forward in time (in large time steps) and checks for convergence to a steady state. By going over each of the options used in this command, we will provide an overview of the operation and capabilities of the `simulate_ode` command. The "`suffix ⇒ equil`" appends "`_equil`" to the basename for output files of the simulation, which becomes here "`egfr_simple_equil`". This prevents output files from the equilibration phase from being overwritten by subsequent simulation commands. The end time (`t_end`) for the simulation is given a sufficiently large value to ensure that steady state is reached prior to the end of the simulation (*see* **Note 36**). The number of steps at which

results are written to the output files is specified by the `n_steps` parameters, which is set to a relatively small value here because we are only interested in reaching steady state and not in tracking the time course. The interval between reporting of results is given by (`t_end/n_steps`), which is 10,000 s in this case. (Note that the `n_steps` parameter controls only the reporting interval and not the step size used by the CVODE solver, which uses adaptive time stepping). Results can also be reported at unevenly spaced intervals (*see* **Note 37**). The `sparse` option invokes fast iterative methods in the CVODE solver that can greatly accelerate the simulations (*see* **Note 38**). The `steady_state` flag causes a check for the convergence of the species population levels to be performed following each report interval, with the propagation terminating if the root mean square of the relative change in the population levels falls below a threshold, which is taken to be 10×`atol`, the absolute integration tolerance. Note that the basic operation of the `simulate_ssa` command is the same as that of the `simulate_ode` command. A summary of options available for the simulation commands is given in **Table 1**. Of the options discussed above, only `steady_state` and `sparse` are not available for use with `simulate_ssa`.

After completion of a simulation, the final population levels of all species in a network are saved and used by default as the initial population levels for subsequent simulation commands. In the example, we have modified or overridden this behavior by using the `setConcentration` (*see* **Note 39**) or `resetConcentrations` commands (*see* **Note 40**). Additional options are discussed in **Subheading 3.6.4**.

3.6.3. Viewing the Simulation Results

We now consider visualization of the output produced by the two simulation commands that follow equilibration. Each simulation is run from the same initial conditions, but the second is run using the `simulate_ssa` command, which produces a stochastic (discrete-event) trajectory using the direct method of Gillespie *(40)*. Trajectory data are written into two multicolumn output files for each simulation: a .gdat file that reports the value of each defined observable at each sample time and a .cdat file that reports the population level of every species in the network at each sample time. To avoid overwriting the data produced by `simulate_ode`, the `simulate_ssa` command sets the suffix parameter to "`ssa`", so that the basename of the file becomes "`egfr_simple_ssa`". Both data file types are in ASCII format, so they can be viewed in a text editor or imported into any number of different plotting and data analysis programs. The BioNetGen distribution includes the PhiBPlot plotting utility, which is a Java program that can be run by double-clicking on the file `PhiBPlot.jar` in the `PhiBPlot` subdirectory of the distribution or by typing "`java –jar path/PhiBPlot.jar [datafile]`"

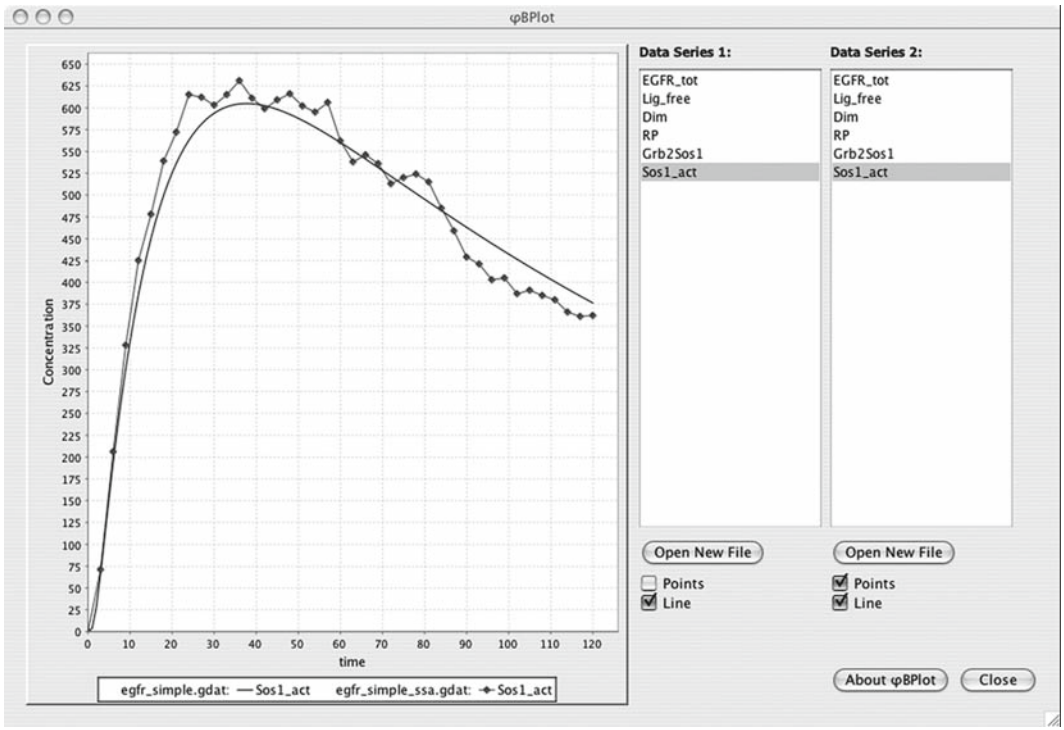

Fig. 3. Plotting BioNetGen simulation data in PhiBPlot. Data from up to two different files may be plotted simultaneously. Here, data for the `Sos1_act` observable from the ODE and SSA simulations is overlaid, showing the effects of fluctuations in the stochastic simulation.

on the command line. PhiBPlot can display data from up to two BioNetGen data files at a time and is useful for quickly visualizing the results of a BioNetGen simulation and for comparing the results of two (*see* **Fig. 3**).

3.6.4. Simulating a Previously-Generated Network

Network generation can be the most time-consuming part of processing a BioNetGen input file, and during repeated simulations of the same network (e.g., with varying parameters) one may wish to avoid regenerating the network. There are several ways to achieve this outcome. The first way, presented in the example above, is to run multiple simulations within the same input file using the `saveConcentrations`, and `resetConcentrations` commands in combination with the `setConcentration` and `setParameter` commands to vary initial conditions and parameters (*see* **Note 41**).

In some cases, however, it may be desirable to reload a network that was generated during a previous invocation of `BNG2.pl`. The `readFile` command provides a way to fully restore a previously generated network so that parameters and species concentrations can be modified using the `set` commands. The basic syntax is illustrated by the command

```
readFile({prefix=>"testread",file=>
"egfr_simple.net"}),
```

which restores the network generated in the example of **Listing 1** with population levels set to their postequilibration values (*see* **Note 42**). The `readFile` command, unlike other BioNetGen commands, resets the global basename to be the basename of the `file` argument, which is "`egfr_simple`" in the example given above. The `prefix` parameter is set here to override this behavior and to set the basename for subsequent simulations commands to "`testread`" rather than `egfr_simple`.

Reading a previously generated network from a file is always much faster than regenerating the network, but can still be time-consuming for very large networks. It may, therefore, be advantageous to pass the previously generated .net file directly to the simulation program by using the `netfile` argument to the `simulate_x` command, as in

```
simulate_ode({netfile=>"egfr_simple.net
t_end=>120,n_steps=>12});
```

The disadvantage of this method is that it does not permit the model parameters to be changed without directly editing the .net file (*see* **Note 43**).

3.7. Additional Examples

In this section, we discuss two example applications of BioNetGen. In the first example, we illustrate how BioNetGen can be used to extend a conventional model so that it can be used to interpret fluorescent labeling experiments. In the second example, we illustrate how BioNetGen can be used to produce a model for a system in which polymerization-like reactions are possible (e.g., a model for multivalent ligand-receptor interactions). The graphical formalism upon which BioNetGen is based was designed with these types of systems in mind *(25)*. The structured objects (graphs) of BNGL allow the topological connectivity of (protein) complexes to be explicitly represented and tracked in a model.

3.7.1. Fluorescent Labeling

Here we illustrate how BioNetGen can be used to extend an existing (nonrule-based) reaction network. In some cases, one needs to add a property that is passed from one species to another in a reaction network. For example, many experiments involve fluorescent labeling, in which the system is injected with fluorescently-labeled proteins that can be monitored. Fluorescent species carry all the properties of nonfluorescent species, but can also be photobleached, losing fluorescence. Given a reaction network of nonfluorescent species, the network that includes both fluorescent and nonfluorescent species nearly doubles in size. For larger networks, this expansion will be error-prone if done manually.

Thus, it is desirable to be able to extend a model to enable tracking of fluorescent labels, and BioNetGen provides such a capability by allowing the definition of a mapping of component state labels from reactants to products. In addition to the application shown here, these mappings have been used to define carbon fate maps for many of the currently known reactions in metabolism *(38)*.

We consider a simple reaction network consisting of five species and described by four basic reactions (considering each direction as a separate reaction)

$$A + B \underset{k_{-AB}}{\overset{k_{+AB}}{\longleftrightarrow}} C$$

$$C + D \underset{k_{-CD}}{\overset{k_{+CD}}{\longleftrightarrow}} E$$

The label chemistry we want to describe works as follows: fluorescence is passed from A to C in reaction 1 and from C to E in reaction 2. This can be described by adding a component, which we will call "`f`", to the molecules A, C, and E. The f component in each molecule may be in either the "`off`" or the "`on`" state, as shown in the `molecule types` definitions of **Listing 2**. We then define rules for mapping the state of the f component between

Listing 2. BioNetGen input file for the fluorescent labeling example **(see Subheading 3.7.1)**.

```
begin parameters
NA 6.02e23 # Avogadro's number (molecues/mol)
f  0.1       # Fraction of the cell to simulate
Vo f*1.0e-10 # Extracellular volume=1/cell_density (L)
V  f*3.0e-12 # Cytoplasmic volume (L)
# Initial concentrations (copies per cell)
A_tot 10000
B_tot  8000
D_tot 50000
# Rate constants
# Divide by NA*V to convert bimolecular rate constants
# from /M/sec to /(molecule/cell)/sec
kpAB 3.0e6/(NA*V)
kmAB 0.06
kpCD 1.0e6/(NA*V)
kmCD 0.06
kpI  1.0e7/(NA*V)
kmI   0.1
end parameters

begin molecule types
A(f~off~on)
B()
C(f~off~on)
D()
E(f~off~on)
I()
```

```
end molecule types

begin seed species
A(f~off) A_tot
B()      B_tot
C(f~off) 0
D()      D_tot
E(f~off) 0
I()      0
end seed species

begin reaction rules
1 A(f%1) + B() <-> C(f%1) kpAB, kmAB
2 C(f%1) + D() <-> E(f%1) kpCD, kmCD
3 A(f~off) + I <-> A(f~on) kpI, kmI
end reaction rules

begin observables
Molecules A_f A(f~on)
Molecules C_f C(f~on)
Molecules E_f E(f~on)
Molecules Tot_f A(f~on) ,C(f~on),E(f~on)
end observables

generate_network({overwrite=>1});
# Equilibrate
simulate_ode({suffix=>equil,t_end=>10000,n_steps=>10,\
  steady_state=>1});
# Add indicator
setConcentration("I","A_tot/10");
simulate_ode({t_end=>200,n_steps=>50});
```

A and C (*see* rule 1 in `reaction rules` block of **Listing 2**) and between C and D (*see* rule 2 in `reaction rules` block of **Listing 2**) using the "`%`" character followed by a string to tag components (*see* **Note 22**). By not specifying the component state of f in the rules, we cause the component state to be mapped from the selected reactant molecule to the created product molecule. This trick allows us to avoid writing separate rules for the labeled and unlabeled species. (When mapping components in this way the user should be careful that the allowed state label values of the components are the same or an error will be generated.) The defined observables track the amount of label associated with each of the molecules that can be labeled (`A_f`, `C_f`, and `E_f`) and the total amount of label present in the system (`Tot_f`). The resulting network has 9 species and 10 reactions.

There are different ways in which labeled components may be introduced into the system. The simplest way would be to define an initial pool of labeled A molecules, i.e., define the species "`A(f~on)`" to have nonzero initial concentration. Here, we have chosen a somewhat more complex scenario in which the system is

initially equilibrated without the label, followed by the introduction of an indicator molecule that adds label to A through a chemical reaction, the third rule in the input file. Following equilibration with no indicator present, the indicator concentration is set to be a fraction of the total number of A molecules using the `setConcentration` command. Results of simulation of the network following equilibration and introduction of the indicator molecule are shown in **Fig. 4**. The labeling reaction (rule 3) is fast compared with the other reactions, so that labeled A initially accumulates followed by a slower rise in the levels of labeled C and D molecules.

3.7.2. Polymerization

BNGL can be used to model the kinetics of molecular aggregates having different topological structures, such as chains, rings, and trees. Here, we present a simple model for the binding of a soluble multivalent ligand to a bivalent cell-surface receptor, such as a membrane-bound antibody. In this model, we consider a

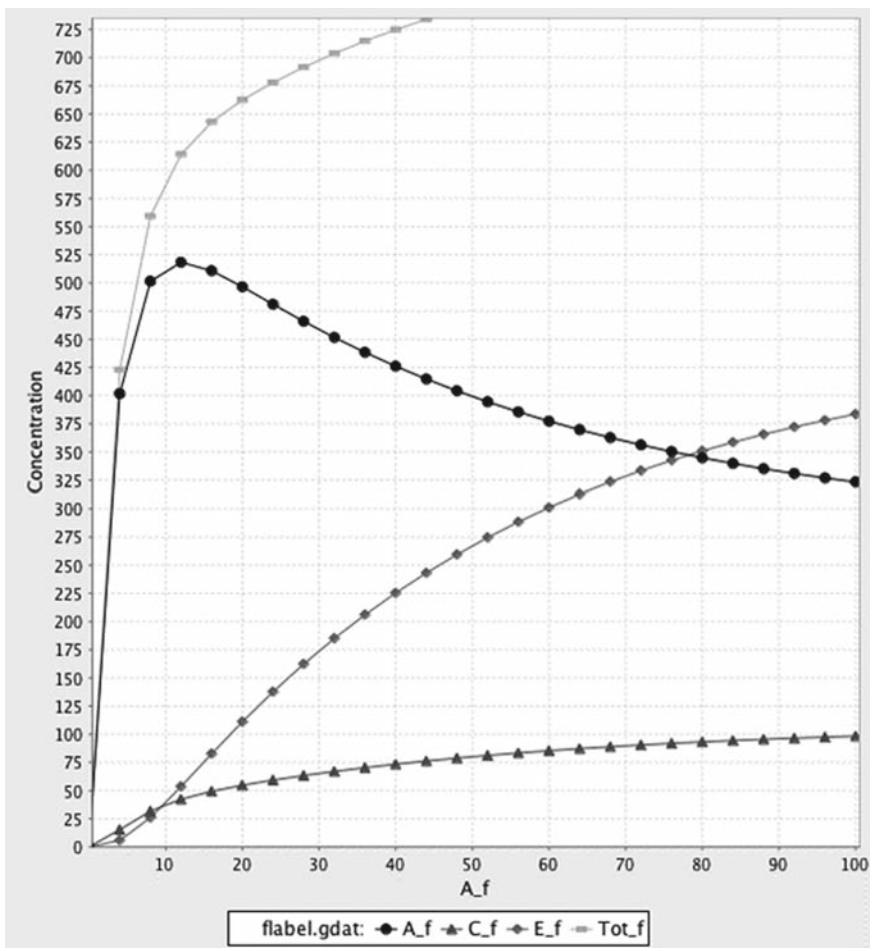

Fig. 4. Plot of simulation results obtained from BioNetGen input file for the fluorescent labeling example shown in **Listing 2** made using PhiBPlot (black and white rendition of color output). The plot shows time courses of the observables from the second `simulate_ode` command in the `actions` block of **Listing 2**.

bivalent ligand with two identical binding sites (`L(l,l)`) and a bivalent receptor with two identical binding sites (`R(r,r)`). The ligand may cross-link two receptors to form a dimeric receptor aggregate (`R(r,r!1).L(l!1,l!2).R(r!2,r)`), which can then interact with additional ligand via free receptor sites. Ligand-receptor interaction can form a distribution of linear chains of alternating ligands and receptors (`R(r,r!1).L(l!1,l!2).R(r!2,r!3).L(l!3,l!4).…`). Two simple rules, shown in **Listing 3**, provide an elementary model of bivalent ligand–bivalent receptor interaction under the assumptions that the length of a chain does not affect its reactivity and that rings do not form (*see* **Note 44**). A third rule that allows the formation of rings of any size is shown in **Listing 3**, but this rule is commented out (*see* **Note 45**). For a different example of polymerization in a biological context, *see* **Note 46**.

Listing 3. BioNetGen input file for binding of bivalent ligand to bivalent receptor (**see Section 3.7.2**).

```
setOption(SpeciesLabel,HNauty);
begin parameters
NA 6.02e23 # Avogadro's number (molecues/mol)
f  0.001       # Fraction of the cell to simulate
Vo f*1.0e-9 # Extracellular volume=1/cell_density (L)
V  f*3.0e-12 # Cytoplasmic volume (L)
L0  1e-9*NA*Vo # Conc. in Molar -> copies per cell
R0  f*3e5
kp1 3.3e/(NA*Vo)
km1 0.1
kp2 1e6/(NA*V)
km2 0.1
kp3 30
km3 0.1
end parameters

begin molecule types
 R(r,r)
 L(l,l)
end molecule types

begin reaction rules
# Ligand addition
1 R(r) + L(l,l) <-> R(r!1).L(l!1,l) kp1,km1
# Chain elongation
2 R(r) + L(l,l!+) <-> R(r!1).L(l!1,l!+) kp2,km2
# Ring closure
#3 R(r).L(l) <-> R(r!1).L(l!1) kp3,km3
end reaction rules
```

```
begin seed species
 R(r,r) R0
 L(l,l) L0
end seed species

begin observables
Species FreeL L(l,l)
Dimers  R==2
Trimers  R==3
4mers   R==4
5mers   R==5
6mers   R==6
7mers   R==7
8mers   R==8
9mers   R==9
10mers  R==10
gt10mers R>10
end observables

# Simulation of a truncated network
generate_network({overwrite=>1,max_stoich=>{R=>10,L=>10}});
simulate_ode({t_end=>50, n_steps=>20});

# Simulation on-the-fly
generate_network({overwrite=>1,max_iter=>1});
simulate_ssa({t_end=>50,n_steps=>20});
```

The `observables` block in **Listing 3** introduces a new syntax for using stoichiometry in the definition of observables, which is needed to track the aggregate size distribution in models that exhibit polymerization (*see* **Note 47**).

Because chains can grow to any length, unless stopping criteria are specified, the process of iterative rule application initiated by a `generate_network` command will not terminate until the user runs out of patience or the computer runs out of memory. We discuss here two methods of simulating a network that cannot be enumerated completely.

The first method is to specify any of the restrictions described in **Subheading 3.6.1** on the `generate_network` command, which will cause termination before all possible species and reactions have been generated. The first pair of actions in **Listing 3** shows how the `max_stoich` parameter can be used to limit the stoichiometry of complexes, producing in this case a network of 30 species and 340 reactions, which can be rapidly simulated using either the ODE or SSA methods. The accuracy of simulations on

artificially truncated networks is, however, not guaranteed and may depend strongly on the parameter values. For the parameters shown in **Listing 3**, the population of clusters with more than about 5 receptors is small, and little error results from network truncation. However, if the value of the cross-linking parameter, `kp2`, is increased by a factor of 10, the cluster size distribution generated by the truncated network becomes inaccurate. The user must therefore be careful to check results for convergence, particularly when changing the parameter values over substantial ranges.

The second method, which is specified by the second pair of actions in **Listing 3**, is to do a minimal initial round of network generation and then allow the network to be generated as new species become populated during a stochastic simulation. The call to `generate_network` is required here to generate the reactions that can take place among the seed species; otherwise, an error will occur when a simulation command is invoked and there are no reactions in the network. With `max_iter` set to 1 only reactions involving seed species are initially generated. During simulation initiated with the `simulate_ssa` command, BioNetGen detects when a reaction event occurs that populates one or more species to which rules have not been previously applied and automatically expands the network through rule application. This behavior is built into the `simulate_ssa` command and no additional parameters need to be specified. The performance of on-the-fly simulation is highly dependent on the system parameters and on the number of molecules being simulated. Increasing the number of molecules while holding the concentrations fixed (accomplished by changing the parameter `f`) increases the size of the network that is generated by on-the-fly sampling. Because the network generation involves the computationally expensive step of generating and comparing canonical labels (*see* **Note 33**), the simulation performance can become poor if one attempts to simulate on-the-fly under conditions that lead to the possible formation of more than about 10^3–10^4 species. Simulation of the dynamics of 300 receptors up to steady state takes about 30 CPU seconds on a MacBook Pro with the 2.4 GHz Intel Core Duo processor and generates a network of about 50 species and 350 reactions.

In the near future, a third and more powerful option will be available for simulating large-scale networks, such as those that arise when polymerization is possible or when some of the signaling molecules have high valence (*see* **Note 48**). Work is currently underway to implement the discrete-event particle-based simulation method that has been recently developed, which extends Gillespie's method to consider rules rather than individual reactions as event generators *(30, 31)*. The main idea behind this method is that by tracking individual particles in a simulation rather than populations the need to explicitly enumerate the possible species and reactions is eliminated. The computational scaling of a stochastic, event-driven simulation

using the particle-based approach becomes effectively independent of network size and has moderate (logarithmic) scaling with the size of the rule set. This rule-based kinetic Monte Carlo method offers significantly better performance than the earlier particle-based event-driven algorithm used in the StochSim software, which uses a less efficient event sampling algorithm that produces a high fraction of nonreactive events *(21)*. The planned incorporation of the rule-based kinetic Monte Carlo method will enable the efficient simulation of comprehensive models of signal transduction networks on the basis of molecular interactions, and, we hope, greatly increase the power of predictive modeling of such systems.

The plot in **Fig. 5** shows simulation results for the number of receptors in trimers as a function of time (in seconds) from the ODE simulation of the truncated network (smooth line) and the SSA simulation with on-the-fly network generation (jagged line). Following the initial equilibration period about 10–20% of the receptors are in trimers at any given time. The total time required for network generation and simulation is comparable in the two cases, with network generation consuming the vast majority of the CPU time.

3.8. Concluding Remarks

The information provided here serves as both an introductory guide and reference resource for the modeler interested in using BioNetGen to develop and analyze rule-based models of bio-

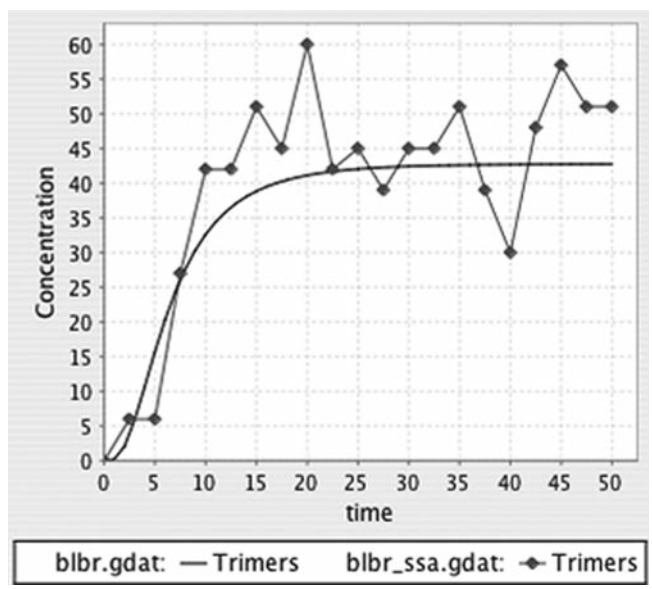

Fig. 5. Plot of simulation results obtained from BioNetGen input file for the bivalent ligand bivalent receptor binding model shown in **Listing 3** made using PhiBPlot. *Smooth solid line* is the curve obtained from the `simulate_ode` command; *jagged line* with *circles* shows results from the `simulate_ssa` command.

chemical systems. Several applications of BioNetGen have been presented and discussed, but the rule-based modeling approach enabled by BioNetGen can be used for a much broader range of purposes. We strive to be responsive to the needs of the BioNetGen user community and encourage users to contact us to share their experiences, to request new capabilities and features, and to report bugs. The BioNetGen web site (http://bionetgen.org) has a wiki format to allow users to contribute information and models. Updates of the information presented here will be announced at the wiki site. Rule-based modeling of biochemical networks is a rapidly evolving area of research and BioNetGen is therefore very much a work in progress, with new capabilities being added continually.

BioNetGen is an open-source project. Although contributions of code are welcome, the main reason the source code is made available is so that users can see how the code works and can confirm that model specifications are being processed as expected. Because of the difficulties of checking the correctness of a chemical reaction network or a simulation result generated automatically from rules, key elements of BioNetGen have been coded independently multiple times and crosschecked. After extensive testing, we are confident that the software is reliable. By following the guidance provided here, a modeler should be able to precisely use BNGL to obtain intended model specifications.

In the future, we hope to see the BioNetGen framework evolve to enable community-driven development of comprehensive models for cellular regulatory systems. The material components and interactions of a cellular regulatory system are typically too numerous and complicated for a single researcher to thoroughly document and capture faithfully in a model of comprehensive scope. For example, nearly 200 proteins are documented to be involved in EGFR signaling in the NetPath database (http://netpath.org). The ability to extend models through the composition of rules is a key factor that makes incremental construction of large-scale models a real possibility *(8, 27)*. To take advantage of collective intelligence for the construction of large-scale models, we are actively pursuing the following extensions of the BioNetGen framework: (1) implementation of methods capable of simulating models composed of a large number of rules *(30, 31)*, (2) manipulation and encoding of BNGL using an XML-based format proposed as an extension of SBML (http://sbml.org) to better facilitate electronic exchange and storage of models, and (3) development of conventions and database-related tools for annotating models and model elements (e.g., linking of molecule names in a model specification to amino acid sequences and other information in standard databases). However, for a long time to come, we foresee that a sound understanding of the material presented here will be useful for rule-based modeling with BioNetGen.

4. Notes

1. Users familiar with a command line interface on any of these systems should have no trouble following the instructions for using the software after reading this chapter. Other users may find the RuleBuilder application, which provides a graphical user interface to BioNetGen, more accessible. This application may be started by double-clicking on the `RuleBuilder-beta-1.51.jar` file in the RuleBuilder subdirectory of the BioNetGen distribution. The RuleBuilder Getting Started Guide in the same directory explains use of the software. Although this chapter focuses on the text-based interface, the basic concepts of BioNetGen modeling discussed here are essential for proper use of RuleBuilder.
2. BioNetGen is invoked in a command shell using `prompt> path/Perl2/BNG2.pl file.bngl`
3. The syntax of a line in the `parameters` block is `[`*`index`*`]` *`parameter`* `[=]` *`value`* where square brackets indicate optional elements, *`parameter`* is a string consisting of only alphanumeric characters plus the underscore character ("_") and containing at least one nonnumeric character. *`value`* may be either a number in integer, decimal, or exponential notation or a formula involving numbers and other parameters in C-style math syntax. *See* **Listing 1** for examples.
4. The size of the system being simulated can be scaled by changing the value of the parameter `f` in **Listing 1**. By scaling all of the initial populations and the volumes by this factor, the system size is scaled without changing the *concentrations* of any of the constituents. For a deterministic simulation, the simulation time and the behavior of the system (e.g., the value of any observable divided by `f`) is independent of `f`. For a stochastic simulation, however, the time required to carry out a simulation will be proportional to `f`, whereas the noise will be proportional to `1/sqrt(f)`.
5. In current BNGL each component may have at most one associated state label, which may take on an arbitrary number of discrete values, specified as strings. The state is thus a scalar variable that can be considered as an enum data type. Future planned extensions of BNGL include nesting of components to allow a single component to have multiple associated states and binding sites.
6. Names for all BioNetGen objects other than parameters, which includes molecules, components, state labels, bonds, labels, and observables may consist of alphanumeric characters and the underscore character ("_"), but may not include the dash character ("-"), which is sometimes used in the biologi-

cal literature as part of protein or domain names. It is not an allowed character here because in some contexts it may be confused with the arithmetic minus operator.

7. The syntax of a line in the `molecule types` block is

   [*index*] *moleculeType*

   where *moleculeType* has the syntax described in the text and illustrated in **Listing 1**.

8. If the `molecule types` block is present, all molecules in the `seed species` block must match the type declarations in the `molecule types` block. A molecule matches its type declaration if each of its declared components is present and each component state is a member of the declared set of possible states. If the `molecule types` block is not present, then the `seed species` block serves a typing purpose. The first instance of a molecule in the `seed species` block is taken to define the complete set of components in that molecule in the model, and only components that are assigned a state in the first occurrence may subsequently have defined states. For example, the Grb2 molecule implicitly defined by the species `Grb2(SH2,SH3)` may not have any states assigned to SH2 or SH3 components. However, the species `EGFR(L,CR1,Y1068~U)` defines the Y1068 component of EGFR as one that has an associated state label, which has at least one allowed value, "`U`", and potentially others to be defined later. Occurrences of additional allowed state labels may occur in the `seed species` block or in the `reaction rules` block, and in either case BioNetGen generates a warning message that additional allowed state values are being associated with the component.

9. The syntax of a line in the seed species block is

   [*index*] *species* [*initialPopulation*]

   where *species* has the syntax for a BioNetGen species as described in the text and illustrated in the `seed species` block of **Listing 1** and *initialPopulation* is a number or formula that specifies the amount of the species present at the start of the first simulation (default is zero).

10. The amount of a chemical species may be specified to have a constant value by prefixing the chemical species name in the `seed species` block with a "$" character, as follows: the expression "`$EGF(R) 1`" would set the amount of free EGF in the system to 1. This feature is useful for considering certain scenarios.

11. The syntax of a line in the `observables` block is

    [*index*] [*observableType*] *observableName* *pattern1*[, *pattern2*]...

    where *observableType* is either `Molecules` or `Species` (defaults to `Molecule` if omitted) *observableName* is a valid name for a BioNetGen observable, and each *pattern* is a valid BioNetGen pattern.

12. Recall that bond names are arbitrary and are used only to identify the bond endpoints. Thus, the bond names used in a pattern do not affect the resulting matches.

13. The "`?`" wildcard can also be used in state matching, but leaving component state out of a match is more commonly achieved by omitting the state label altogether. For example the patterns "`EGFR(Y1068)`" and "`EGFR(Y1068~?)`" are equivalent, i.e., generate the same matches.

14. For each pattern, selected matches to species in the model of **Listing 1** are listed with the image of the pattern elements shown in **bold**. (These are not meant to be exhaustive, just illustrative.) Note that some chemical species are matched multiple times by a given pattern.

    a. **`EGFR()`** matches
    `EGFR``(CR1,L,Y1068~U),EGF(R!1).`
    `EGFR``(CR1,L!1,Y1068~U), EGF(R!1). EGF(R!2).`
    `EGFR``(CR1!3,L!1,Y1068~U).EGFR(CR1!3,`
    `L!2,Y1068~U),` and `EGF(R!1).EGF(R!2).EGFR(CR`
    `1!3,L!1,Y1068~U).`**`EGFR`**`(CR1!3,L!2,Y1068~U)`

    b. **`EGF(R)`** `matches` **`EGF(R)`**

    c. **`EGFR(CR1!+)`** matches `EGF(R!1).EGF(R!2).`
    `EGFR``(`**`CR1!3`**`,L!1,Y1068~U).`
    `EGFR(CR1!3,L!2,Y1068~U), and EGF(R!1).`
    `EGF(R!2).EGFR(CR1!3,L!1,Y1068~P).`
    `EGFR(CR1!3``,L!2,Y1068~U`**`)`**

    d. **`EGFR(Y1068~P!?)`** matches `EGF(R!1).`
    `EGF(R!2).`**`EGFR`**`(CR1!3,L!1,`**`Y1068~P`**`).`
    `EGFR(CR1!3,L!2,Y1068~U), and EGF(R!1).`
    `EGF(R!2).`**`EGFR`**`(CR1!3,L!1,`**`Y1068~P!4`**`).`
    `EGFR(CR1!3,L!2,Y1068~U).Grb2(SH2!4,SH3)`

    e. **`Grb2(SH2,SH3!1).Sos1(PxxP!1)`** matches
    `Grb2(SH2,SH3!1).Sos1(PxxP!1)`

    f. **`EGFR(Y1068!1).Grb2(SH2!1,SH3!2).`**
    `Sos1(PxxP!2)` matches `EGF(R!1).EGF(R!2).`
    `EGFR``(CR1!3,L!1,`**`Y1068~P!4`**`).EGFR(CR1!3,L!2,`
    `Y1068~U).`**`Grb2(SH2!4,SH3!5).Sos1(PxxP!5)`**

15. The sum corresponding to an observable is defined explicitly in the .net file that is generated by BioNetGen when an input file is processed. These sums are contained in the `groups` block of the .net file (*see* **Note 28**).

16. When an observable is defined by two or more patterns, the associated functions are computed as follows. For an observable of the Molecules type, the observable is a sum of the observables defined by each individual pattern in the set. For an observable of the Species type, the observable is an unweighted sum of the populations of chemical species matched by any of the patterns in the set. Multiple patterns can be useful for specifying observables that are functions of multiple sites on a molecule, e.g., the total phosphorylation level of a protein that can be phosphorylated at multiple sites.
17. The .gdat and .cdat files produced by BioNetGen simulation commands are ASCII text files that list the time courses of observables and concentrations, respectively, in a tabular format. The first line of each file is a header beginning with a "#" character, followed by a whitespace-separated list of strings identifying the contents of each column. The first column is "time" in both .gdat and .cdat formats. In a .gdat file the remaining columns list the observable names corresponding to each column. In the .cdat file, the remaining columns list the index of the species concentration corresponding to each column.
18. The syntax of a line in the `reaction rules` block is

    [*index*] *rPattern1* [*+rpattern2*] … *arrow pPattern1* [*+pPattern2*] … *rateLaw1*[*,rateLaw2*] [*command1*]…

    where each *Pattern* is a valid BioNetGen pattern, arrow is one of "–>" or "<–>," each *rateLaw* is a parameter or a rate law function (*see* **Note 20**), and commands have the syntax described in **Subheading 3.5.7**.
19. If a component of a molecule appears in a reactant pattern, the corresponding molecule in the product pattern, if it is not deleted, must include that component. Failure to include the full set of components referenced by the reactant pattern will produce an error. Thus, the rule "`A(a)-> A(b) kab`" produces an error, even if the A molecule has both components a and b.
20. Other rate laws are invoked by using one of the keywords for the allowed rate law types followed by a comma-separated list of numerical values or formulas in parentheses. As of this writing, the three recognized rate law types are "`Ele`", "`Sat`", and "`MM`". The formula for the Ele rate law is

$$\mathrm{Ele}(k_1) = k_1 \prod_{i=1}^{M} x_i ,$$

    where M is the molecularity of the reaction (i.e., the number of reactants) and xi is the population level of the ith reactant.

This rate law type is specified by default when only a numerical value or a formula is given following the product patterns in a rule, as described in Subheading 3.5. The current version of BioNetGen supports a few nonelementary rate law formulas, primarily to allow simulation of models from the literature that incorporate these rate laws. The formula for the Sat rate law is

$$\mathrm{Sat}(k_{\mathrm{cat}}, K_{\mathrm{m}}) = k_{\mathrm{cat}} \prod_{i=1}^{M} x_i \,/\, (K_{\mathrm{m}} + x_1)\,,$$

where x_1 is the population level of the reactant matching the first reactant pattern in the rule. Note that $V_{\max} = k_{\mathrm{cat}} x_2$ and K_{m} are the usual Michaelis–Menten parameters if M= 2 *(47)* and that these parameters should be specified in consistent units. An example of a rule that uses this rate law is

```
Prot(Y ~ U) + Kinase(aloopY ~ P)®Prot(Y ~ P) +
                        e(aloopY ~ P) Sat(kcat,Km)
```

The formula for the MM rate law is

$$MM(k_{\mathrm{cat}}, K_{\mathrm{m}}) = k_{\mathrm{cat}} x'_1 x_2 \,/\, (K_{\mathrm{m}} + x'_1)\,,$$

where $x'_1 = \left((x_1 + x_2 - K_{\mathrm{m}}) + \sqrt{(x_1 + x_2 - K_{\mathrm{m}})^2 + 4K_{\mathrm{m}} x_1}\right) / 2$.

Note that this rate law type is applicable only if M = 2. The MM rate law type is the same as the Sat rate law type when M=2 except that x_1 is replaced by x'_1 to account for the amount of "substrate" bound to "enzyme." In the near future it will be possible to define rate laws using arbitrary mathematical formulas.

21. A single-site rate law characterizes the rate of a reaction that involves the formation or dissolution of a single bond. In some cases, a reaction can occur in multiple ways that are indistinguishable. In these cases, the single-site rate law needs to be multiplied by a statistical factor to obtain the appropriate observable rate of the reaction. For example, if an antibody with two identical binding sites associates with a monovalent hapten, then there are two indistinguishable ways that this reaction could occur. If the single-site rate constant is k, then the observable rate at which the reaction occurs is 2 k [IgG] [hapten], where [IgG] is the concentration of bivalent antibody, [hapten] is the concentration of monovalent hapten, and the statistical factor of 2 accounts for the fact that hapten can add to either of the two sites on the antibody. BioNetGen in generating or simulating a reaction network automatically accounts for such statisti-

cal factors under the assumption that the rate law associated with a rule applies to a single-site reaction. A modeler should therefore be careful to always specify a single-site rate constant when writing a rule. Likewise, BioNetGen automatically adds a symmetry factor of $1/2$ to account for reactions such as A + A $\rightarrow$ product(s), a factor of $1/6$ to account for reactions such as A + A + A $\rightarrow$ product(s), etc. In general, when assigning a rate constant to the elementary rate law of a rule, one should assign the constant appropriate for a reaction of the form A+B $\rightarrow$ product(s) where in this reaction there is a unique path from the reactants to product(s). BioNetGen will automatically correct rates of reactions for statistical and symmetry factors. This feature is important because these factors often vary from reaction to reaction within a class of reactions defined by a single rule *(28)*.

22. Any component in a reaction rule may be tagged by adding the "`%`" character followed by the tag name. The scope of a tag is local to the rule in which it appears.

23. The application of rule 1 of **Listing 1** to the species {**`EGF(R), EGFR(L,CR1`**`,Y1068~U)`**`, EGFR(L,CR1`**`,Y1068~P)`**`, EGFR(L,CR1`**`,Y1068~P!1).Grb2(SH2!1,SH3)}`, produces the following reactions:

```
EGF(R)+EGFR(L,CR1,Y1068~U)->
EGF(R!1).EGFR(L!1,CR1,Y1068~U) kp1
EGF(R)+EGFR(L,CR1,Y1068~P)->
EGF(R!1).EGFR(L!1,CR1,Y1068~P) kp1
EGF(R)+EGFR(L,CR1,Y1068~P!1).Grb2(SH2!1,SH3)->\
EGF(R!2).EGFR(L!2,CR1,Y1068~P!1).Grb2(SH2!1,SH3) kp1
```

where the images of the reactant patterns are shown in bold and the reaction centers are underlined. The rate law for an individual reaction has the same format as a rate law in a reaction rule (*see* **Note 18**).

24. The scope of a bond name is restricted to the pattern in which it appears. Bond names are not used in establishing the correspondence between reactant and product patterns. Thus, the rule "`A(a!1).B(b!1~U) -> A(a!2).B(b!2~P)`" has no effect on the bond between A and B even though in the specification the name of the bond changes between the reactant and product sides. Similarly, in the expression "`A(a!1).B(b!1) + C(c!1).D(d!1)`" the fact that both bonds have the same name has no consequence.

25. Internally, BioNetGen represents all reactions generated by rules as unidirectional and maintains this representation when generating a .net file or exporting networks to SBML and MATLAB M-file formats.

26. Consider the action of the following two rules on the initial species "`A(a1!1,a2!2).B(b1!1,b2!2)`", which describes a complex between A and B molecules connected by two bonds. Both rules break the bond between the a1 component of A and the b1 component of B. The first rule, "`A(a1!1).B(b1!1) -> A(a1)+B(b1)`," has a molecularity of two in the products, and thus does not apply to this complex because breaking the bond still leaves the complex held together by the bond between a2 and b2. The second rule, "`A(a1!1).B(b1!1) -> A(a1).B(b1)`", does not require dissociation of the resulting complex and generates the reaction "`A(a1!1,a2!2).B(b1!1,b2!2) -> A(a1,a2!1).B(b1,b2!1)`".

27. Describing dephosphorylation as a first order reaction involving only the substrate assumes that the responsible enzyme, a phosphatase is constitutively active and is present at an excess and unchanging level. Dephosphoryation reactions have been handled this way (*see*, e.g., **ref.** *33)* because the identities of the phosphatases acting on a particular substrate are often unknown.

28. The two products are

    `EGF(R!1). EGF(R!2). EGFR(CR1!3,L!1,Y1068~U).`
    `EGFR(CR1!3,L!2,Y1068~P)`

    `EGF(R!1).EGF(R!2).EGFR(CR1!3,L!1,Y1068~P).`
    `EGFR(CR1!3,L!2,Y1068~U)`, which are isomorphic, as can be verified by switching the order of the two EGF and two EGFR molecules and renumbering bonds 1 and 2.

29. A correction is required for *rules* that are symmetric *(26)*. BioNetGen automatically detects rule symmetry generates reactions with the correct multiplicity. Consider the symmetric rule "`A(a) + A(a) → A(a!1).A(a!1) k`" applied to the set of species `{A(a,Y~U), A(a,Y~P)}`. The following reactions will be generated

    ```
    A(a,Y~U) + A(a,Y~U) → A(a!1,Y~U).A(a!1,Y~U)
    0.5*k
    A(a,Y~U) + A(a,Y~P) → A(a!1,Y~U).A(a!1,Y~P) k
    A(a,Y~P) + A(a,Y~P) → A(a!1,Y~P).A(a!1,Y~P)
    0.5*k
    ```

    where the first and third reactions are symmetric and thus have a multiplicity of ½, for the reason discussed above in **Note 21**. The second reaction has a multiplicity of 1 because there is only one way that a bond may be added to join the two A molecules.

30. This would not be the case if EGFR had another binding partner that could bind through the CR1 domain, such as another member of the ErbB family of receptors to which

EGFR belongs. Consider a simplified example of the protein "A(Y~U,b)", where the b component is a binding site that can bind either to the b site of a kinase B or to the b site of a kinase-dead mutant of B called Bi. The rule "A(Y~U,b!+)->A(Y~P,b!+)" would not capture the described mechanism because both **A(Y~U,b!1)**.B(b!1) and **A(Y~U,b!1).**Bi(b!1) would generate phosphorylation reactions under action of this rule. The most obvious way to address this problem is to explicitly specify that B must be present for the rule to apply, as in "A(Y~U,b!1).B(b!1)->A(Y~P,b!1).B(b!1)".

31. Such counters can prevent the steady_state flag of simulate_ode from reaching steady state because the counter species will increase linearly in time if the steady-state concentration of the A-containing species is nonzero. If one wishes to preserve possible steady-state behavior, the concentration of the Trash species should be fixed by prepending a "$" character to its declaration in the seed species block (*see* **Note 10**). In other words, Trash should be declared as a seed species with fixed value using the line "$Trash 0" in the seed species block.

32. Species are compared during network generation by generating a string label for the species from a canonical ordering of the molecules, components, and edges *(26)*. A canonical ordering is one that guarantees that two graphs will generate the same label if and only if they are isomorphic *(48)*. In this way, the problem of determining graph isomorphism is reduced to string comparison and testing a species found in a new reaction for isomorphism with existing species is reduced to looking up its label in a hash table. For labeled graphs, such as those used in BioNetGen to represent species, the problem of canonical ordering is trivial if all labels in the graph are unique (lexical sorting will suffice). More powerful methods are needed if there are multiple occurrences of nodes with identical labels *(49)*. There are three different methods that BioNetGen can use to generate canonical labels and test species for isomorphism. To specify a canonical labeling method the command "setOption("SpeciesLabel",*method*);" is placed anywhere in the BNGL file outside of the input blocks and before the first action command. In this command *method* is Auto, HNauty, or Quasi. Use of this command is optional unless overriding the default method, which is Auto. The default method used by BioNetGen for generating canonical labels is called "Auto," which works by generating a quasi-canonical label that includes all information about the Species except the bonds, for which only the bond order of each Component is listed. These quasi-canonical labels are quick to generate, but they cannot distinguish all nonisomorphic species. Thus,

any two species that share a quasi-canonical label must be further checked for isomorphism directly, for which BioNetGen uses a variant of the Ullmann algorithm *(50)*. This method is always correct, but may be very slow if the number of identical Molecules or Components in a complex is greater than a handful, because it requires checking of a large number of permutations. A second exact method called "HNauty" is available that gives more robust performance when species are formed that involve substantial numbers of repeated elements. HNauty is a generalization of the Nauty algorithm of McKay *(49)* developed specifically to handle graphs representing species in BioNetGen *(51)*. HNauty is slower than Auto when most of the species in a network have low stoichiometry, but is sometimes required to simulate networks in which substantial oligomerization occurs. In some cases, such as when oligomers are restricted to linear chains, the quasi-canonical strings used as a filter by the Auto method turn out to be canonical. If that is the case, the "Quasi" method can be used to turn off the additional isomorphism check for species that match an existing quasi-canonical label, which can significantly accelerate network generation. This method should only be used when the user can confirm that the quasi-canonical labels are in fact canonical; otherwise, failure to resolve nonidentical species will result in unpredictable behavior.

33. If a rule set implies a large or unbounded network and a user attempts to generate the network, BioNetGen may not complete execution in a reasonable amount of time. In such cases a user has several options: (1) Restrict network generation using arguments to `generate_network`, as discussed in **Subheading 3.6**; (2) Use the "`print_iter 1`" option of the `generate network` command to cause BioNetGen to dump an intermediate .net file for each iteration of rule application and inspect the resulting .net file for indications of runaway polymerization that may be unintended; (3) Run a stochastic simulation with on-the-fly network generation (*see* **Subheading 3.7.2**); (4) Wait for the network-free simulation engine(s) to become available (*see* **Fig. 2**). (5) Use the macro model reduction module for BioNetGen, which uses the algorithms described in *(52–54)* to reduce the size of the network that needs to be generated to calculate the specified observables. The module is included in BioNetGen distributions 2.0.47 and later, and is invoked using the command "`MacroBNG2.pl --macro file.bngl`".

34. The .net file produced by BioNetGen is a BNGL file with the three additional blocks `species`, `reactions`, and `groups`, which contain the species, reactions, and observable function definitions that result from network generation. The syntax for the `species` block is identical to that

of the `seed species` block in the BNGL file. It contains a complete list of the species in the network and their concentrations at the current time. The syntax for each line in the `reactions` block is

```
index reactantListproductList [multiplicity*] rateLaw
```

where the *`reactantList`* and *`productList`* are comma-separated lists referring to species by index, *`multiplicity`* is an optional factor multiplying the rate law, and *`rateLaw`* is either a single parameter (for an elementary type) or one of the additional types (*see* **Note 20**). An example of a reaction entry is

```
1 1,7 8 2*kp1
```

which specifies that species 1 and 7 undergo a bimolecular association to produce species 8 with an elementary rate law governed by the rate constant 2*kp1. The syntax for each line in the observables groups block is

```
index group Name speciesList
```

where the *`speciesList`* is a comma-separated list of species indices, each element of which has the form *`[weight*]speciesIndex`*. An example of a sum definition for observable 6 of the example model in **Listing 1** is

```
6 Sos1_act 13,16,18,20,22,2*23
```

In the sum, the population level of Species 23 has a weight of two, whereas the population levels of all other species have the default weight of one.

35. The `prefix` command sets the basename to be the value of its argument, whereas the `suffix` command appends its argument to the basename. For example, the command "`prefix⇒test`" would set the basename to `test`, and the command "`suffix⇒test`" would append "`_test`" to the basename. The scope of changes to the basename is local to the command in which the `prefix` or `suffix` commands appear. The basename for subsequent commands reverts to the basename of the file unless overriden by additional commands. The sole exception to this is the `readFile` action, which sets the global basename to either the value of the `prefix` command, if present, or to the basename of the `file` command.
36. In practice, a modeler should be careful to check by trial and error that `t_end` is sufficiently large to reach steady state. Recall that BioNetGen expects model parameters and variables to have consistent units, so times specified in simulation commands (e.g., by assigning a value to the `t_end` parameter) should be given in units consistent with those

of rate constants, which have units of inverse seconds in all examples presented here.

37. In addition to reporting simulation output at evenly spaced intervals, as specified using the `t_end` and `n_steps` parameters, BioNetGen can also report results at any set of times specified in the `sample_times` array. When this option is used, values of the `t_end` and `n_steps` parameters should not be specified. An example of nonuniform time sampling specified in this way is the command

    ```
    simulate_ode({sample_times [1,10,100]});
    ```

    which will result in observables and species concentrations being reported at $t = 0$ (the start time), 1, 10, and 100.

38. Rule-based networks tend to be sparse, that is, the vast majority of elements of the Jacobian matrix are zero (the elements of this matrix are $J_{ij} = {}^{\partial f_i(\mathbf{x})}/_{\partial x_j}$, where $\dot{x}_i = fi(x)$ is the ODE describing the kinetics of species i). This may not be the case for networks involving extensive oligomerization. Empirically, we have found that networks with more than a few hundred species tend to be accelerated by the use of sparse methods, with major gains occurring for networks of thousands to tens of thousands of species. The largest network that has been simulated with BioNetGen has about 50,000 species and 100,000 reactions. Above that point, the 2 gigabytes of memory addressable on 32 bit architectures is exceeded.

39. The `setConcentration` command has the syntax

    `setConcentration(`*`species,value`*`)`

    where *`species`* is a valid BioNetGen species specification (*see* **Subheading 3.3**) and *`value`* is a number or formula.

40. Note that if the initial concentration of a species is set to a parameter or a formula, changing the value of the parameter or of parameters in the formula using the `setParameter` after the first simulation is run will not affect the species concentrations, which are overwritten following the completion of the simulation.

41. It is straightforward to write scripts that utilize these commands to automate such tasks as parameter scans or averaging multiple stochastic simulations. The Perl script `scan_var.pl`, which is provided in the Perl2 directory of the BioNetGen distrbution, provides a simple example that can be used for scanning the value of a single parameter and could be easily extended to perform more complex actions.

42. A .net file with the name "*`basename`*`.net`" is automatically generated prior to execution of any simulation command and is read by the `run_network` program, which is executed as

a separate process. If a .net file with the same name already exists, it is overwritten. If multiple simulation commands are given in the same input file, it may be useful to use a different basename for each (using either the `prefix` or `suffix` commands), so that the input network to each simulation can be inspected later for information and debugging purposes. In the example shown in **Listing 1**, the first `simulate_ode` command produces the file "`egfr_simple_equil.net`", the second `simulate_ode` command produces the file "`egfr_simple.net`", and the `simulate_ssa` produces the file "`egfr_simple_ssa.net`".

43. BioNetGen's simulation engine, Network, has a command line interface that can be used directly, bypassing `BNG2.pl` altogether. Details of this interface are provided in the source code of `run_network.c`, which is located in the Network2 subdirectory of the distribution. A summary is provided by running the appropriate `run_network` executable in the `bin` directory of the distribution. In addition, `BNG2.pl` outputs the exact command used to execute `run_network` following the tag "`full_command:`"

44. Note that a nearly identical network of species and reactions can be generated by the single rule "`R(r)+L(r)<-> R(r!1).L(r!1) kp1,km1,`" where the difference is that in the single-rule network all reactions will have the same rate constant. This difference is important physically, because ligand that is bound to receptor is restricted to diffuse on the surface of the cell, whereas free ligand diffuses freely in three dimensions. Although restriction to the cell surface decrease the diffusion constant of the ligand, the effective concentration of receptor binding sites greatly increases resulting in a strong enhancement of the ligand-receptor binding rate *(55)*.

45. Rule 3 of **Listing 3** is the simplest ring closure rule that can be specified for this system, and permits the formation of all possible rings in this system, including a monomeric ring in which a single receptor binds the same ligand twice. To exclude this possibility, which may be sterically unfavorable, one could extend the rule to read "`L(l!1).R(r!1,r).L(l) <-> L(l!1).R(r!1,r!2).L(l!2) kp3,km3`," which forces the ring closure to involve a ligand molecule other than the one to which the R molecule is bound. It is also possible to restrict the range of chain sizes that can undergo ring closure by explicitly including all of the molecules that form the ring, or by using a combination of `include_reactants` and `exclude_reactants` commands. For example, the adding the commands "`include_reactants(1,R.R)`" and "`exclude_reactants(1,R.R.R.R)`" to either ring closure rule would only allow the formation of rings containing two or three R molecules. This is desirable from a biophysical

perspective because the rate of ring closure may depend on the distance between the two endpoints of the chain that are being connected *(56)*. In the future, it will be possible to specify such relationships in a single rule using rate laws that depend on the specific properties of a species matched by a pattern in a rule.

46. Actin, which is one of the major components of the cytoskeleton, forms branched structures that play a critical role in many cellular processes including motility *(57)*. A simple model for the formation of branched actin structures is given by the definition of an actin molecule as "`A(b,p,br)`," where the components b, p, and br represent the barbed end, the pointed end and the branching sites of actin respectively, a rule for chain elongation "`A(b)+A(p)<-> A(b!1).A(p!1) kp1,km1`", and a rule for chain branching "`A(br)+A(p)<-> A(br!1).A(p!1) kp2,km2`." The first rule generates linear filaments of actin, which become branched through the action of the second rule. Filaments may be extended either through the addition of monomers or by combination with another filament.

47. The basic syntax is "*`molecule op number`*," where *`molecule`* is a molecule name, *`op`* is one of the comparison operators "`==`," "`<`," "`>`," "`<=`," or "`>=`," and *`number`* is a non-negative integer. This allows the stoichiometry of a single molecule type within a complex to be selected. If the observable is of type `Molecules` (the default), the observable will reflect the total number of molecules in species matching the selected stoichiometry. If the observable is of type `Species`, the observable will reflect the total population of species matching the selected stoichiometry. The current syntax allows stoichiometry of only a single molecule type to be considered at a time.

48. In our experience, the combinatorial explosion becomes a major bottleneck to generating and simulating networks in any realistic model that considers more than a handful of components. A recent model of EGFR signaling by Danos et al. *(27)* provides an example. The model considers 13 proteins, a small subset of the proteins that are active in EGFR signaling, and is composed of 70 rules that generate about 10^{23} species. Other rule-based models of growth factor signaling have produced similar eye-popping numbers *(58, 59)*. Even models that consider a few components may exhibit polymerization. For example, a simple model of Shp2 regulation constructed in BioNetGen involves only two molecule types, and yet must be truncated because the combination of binding and enzyme–substrate interactions generates infinite chains *(36)*. For the trivalent ligand bivalent receptor problem described in Yang et al. *(31)* there is a phase transition in which nearly all of the receptors coalesce into a single giant aggregate, which makes accurate truncation of the network effectively impossible.

Acknowledgments

Work on BioNetGen has been supported by NIH grants GM035556, RR18754, and GM76570 and DOE contract DE-AC52-06NA25396. J.R.F. also acknowledges support from the Department of Computational Biology at the University of Pittsburgh School of Medicine. Integration of BioNetGen into the Virtual Cell was supported by U54 RR022232 NIH-Roadmap grant for Technology Centers for Networks and Pathways. Special thanks to Byron Goldstein for the initial impetus that led to the development of BioNetGen and for his active and ongoing support. We thank the many people who have contributed to the development of BioNetGen and BioNetGen-compatible tools, including Jordan Atlas, Nikolay Borisov, Alexander Chistopolsky, Joshua Colvin, Thierry Emonet, Sarah Faeder, Leigh Fanning, Matthew Fricke, Bin Hu, Jeremy Kozdon, Mikhail Kravchenko, Nathan Lemons, Michael Monine, Fangping Mu, Ambarish Nag, Richard Posner, Amitabh Trehan, Robert Seletsky, Michael Sneddon, and Jin Yang. We also thank Gary An, Dipak Barua, Marc Birtwistle, James Cavenaugh, Ed Clarke, Vincent Danos, Jerome Feret, Andrew Finney, Walter Fontana, Leonard Harris, Jason Haugh, Michael Hucka, Sumit Jha, Jean Krivine, Chris Langmead, Paul Loriaux, Boris Kholodenko, Michael Saelim, Ed Stites, Ty Thomson, and Aileen Vandenberg for their helpful discussions and input. People contributing to the integration of BioNetGen with the Virtual Cell include James Schaff, Ion Moraru, Anuradha Lakshminarayana, Fei Gao, and Leslie Loew.

References

1. Blinov, M. L., Faeder, J. R., Goldstein, B., and Hlavacek, W. S. (2004) BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics* 20, 3289–3291.
2. Kholodenko, B. N. (2006) Cell-signalling dynamics in time and space. *Nat. Rev. Mol. Cell Biol.* 7, 165–176.
3. Aldridge, B. B., Burke, J. M., Lauffenburger, D. A., and Sorger, P. K. (2006) Physicochemical modelling of cell signalling pathways. *Nat. Cell Biol.* 8, 1195–1203.
4. Dueber, J. E., Yeh, B. J., Bhattacharyya, R. P., and Lim, W. A. (2004) Rewiring cell signaling: the logic and plasticity of eukaryotic protein circuitry. *Curr. Opin. Struct. Biol.* 14, 690–699.
5. Pawson, T. and Linding, R. (2005) Synthetic modular systems – Reverse engineering of signal transduction. *FEBS Lett.* 579, 1808–1814.
6. Bashor, C. J., Helman, N. C., Yan, S., and Lim, W. A. (2008) Using engineered scaffold interactions to reshape MAP kinase pathway signaling dynamics. *Science* 319, 1539–1543.
7. Hlavacek, W. S., Faeder, J. R., Blinov, M. L., Perelson, A. S., and Goldstein, B. (2003) The complexity of complexes in signal transduction. *Biotechnol. Bioeng.* 84, 783–794.
8. Hlavacek, W. S., Faeder, J. R., Blinov, M. L., Posner, R. G., Hucka, M., and Fontana, W. (2006) Rules for modeling signal-transduction systems. *Sci. STKE* 2006, re6.
9. Gomperts, B. D., Kramer, I. M., and Tatham, P. E. R. (2003) *Signal Transduction*. Elsevier Academic Press, San Diego, CA.
10. Hunter, T. (2000) Signaling: 2000 and beyond. *Cell* 100, 113–127.
11. Cambier, J. C. (1995) Antigen and Fc receptor signaling: The awesome power of the

immunoreceptor tyrosine-based activation motif (ITAM). *J. Immunol.* 155, 3281–3285.

12. Pawson, T. and Nash, P. (2003) Assembly of cell regulatory systems through protein interaction domains. *Science* 300, 445–452.
13. Pawson, T. (2004) Specificity in signal transduction: From phosphotyrosine-SH2 domain interactions to complex cellular systems. *Cell* 116, 191–203.
14. Seet, B. T., Dikic, I., Zhou, M. M., and Pawson, T. (2006) Reading protein modifications with interaction domains. *Nat. Rev. Mol. Cell Biol.* 7, 473–483.
15. Mathivanan, S., Periaswamy, B., Gandhi, T. K. B., Kandasamy, K., Suresh, S., Mohmood, R., Ramachandra, Y. L., and Pandey, A. (2006) An evaluation of human protein-protein interaction data in the public domain. *BMC Bioinformatics* 7, S19.
16. Mathivanan, S., Ahmed, M., Ahn, N. G., Alexandre, H., Amanchy, R., Andrews, P. C., Bader, J. S., Balgley, B. M., Bantscheff, M., Bennett, K. L., et al. (2008) Human Proteinpedia enables sharing of human protein data. *Nat. Biotechnol.* 26, 164–167.
17. Ong, S. E. and Mann, M. (2005) Mass spectrometry-based proteomics turns quantitative. *Nat. Chem. Biol.* 1, 252–262.
18. Olsen, J. V., Blagoev, B., Gnad, F., Macek, B., Kumar, C., Mortensen, P., and Mann, M. (2006) Global, in vivo, and site-specific phosphorylation dynamics in signaling networks. *Cell* 127, 635–648.
19. Kholodenko, B. N., Demin, O. V., Moehren, G., and Hoek, J. B. (1999) Quantification of short term signaling by the epidermal growth factor receptor. *J. Biol. Chem.* 274, 30169–30181.
20. Schoeberl, B., Eichler-Jonsson, C., Gilles, E. D., and Muller, G. (2002) Computational modeling of the dynamics of the MAP kinase cascade activated by surface and internalized EGF receptors. *Nat. Biotechnol.* 20, 370–375.
21. Morton-Firth, C. J. and Bray, D. (1998) Predicting temporal fluctuations in an intracellular signalling pathway. *J. Theor. Biol.* 192, 117–128.
22. Endy, D. and Brent, R. (2001) Modelling cellular behaviour. *Nature* 409, 391–395.
23. Jorissen, R. N., Walker, F., Pouliot, N., Garrett, T. P. J., Ward, C. W., and Burgess, A. W. (2003) Epidermal growth factor receptor: Mechanisms of activation and signalling. *Exp. Cell Res.* 284, 31–53.
24. Danos, V. and Laneve, C. (2004) Formal molecular biology. *Theor. Comput. Sci.* 325, 69–110.
25. Faeder, J. R., Blinov, M. L., and Hlavacek, W. S. (2005) Graphical rule-based representation of signal-transduction networks, in *SAC '05: Proc. ACM Symp. Appl. Computing*, ACM, New York, NY, pp. 133–140.
26. Blinov, M. L., Yang, J., Faeder, J. R., and Hlavacek, W. S. (2006) Graph theory for rule-based modeling of biochemical networks. *Lect. Notes Comput. Sci.* 4230, 89–106.
27. Danos, V., Feret, J., Fontana, W., Harmer, R., and Krivine, J. (2007) Rule-based modelling of cellular signalling. *Lect. Notes Comput. Sci.* 4703, 17–41.
28. Faeder, J. R., Blinov, M. L., Goldstein, B., and Hlavacek, W. S. (2005) Rule-based modeling of biochemical networks. *Complexity* 10, 22–41.
29. Lok, L. and Brent, R. (2005) Automatic generation of cellular networks with Moleculizer 1.0. *Nat. Biotechnol.* 23, 131–36.
30. Danos, V., Feret, J., Fontana, W., and Krivine, J. (2007) Scalable simulation of cellular signalling networks. *Lect. Notes Comput. Sci.* 4807, 139–157.
31. Yang, J., Monine, M. I., Faeder, J. R., and Hlavacek, W. S. (2007) Kinetic Monte Carlo method for rule-based modeling of biochemical networks. *arXiv:0712.3773.*
32. Goldstein, B., Faeder, J. R., Hlavacek, W. S., Blinov, M. L., Redondo, A., and Wofsy, C. (2002) Modeling the early signaling events mediated by FceRI. *Mol. Immunol.* 38, 1213–1219.
33. Faeder, J. R., Hlavacek, W. S., Reischl, I., Blinov, M. L., Metzger, H., Redondo, A., Wofsy, C., and Goldstein, B. (2003) Investigation of early events in FceRI-mediated signaling using a detailed mathematical model. *J. Immunol.* 170, 3769–3781.
34. Faeder, J. R., Blinov, M. L., Goldstein, B., and Hlavacek, W. S. (2005) Combinatorial complexity and dynamical restriction of network flows in signal transduction. *Syst. Biol.* 2, 5–15.
35. Blinov, M. L., Faeder, J. R., Goldstein, B., and Hlavacek, W. S. (2006) A network model of early events in epidermal growth factor receptor signaling that accounts for combinatorial complexity. *BioSystems* 83, 136–151.
36. Barua, D., Faeder, J. R., and Haugh, J. M. (2007) Structure-based kinetic models of modular signaling protein function: focus on Shp2. *Biophys. J.* 92, 2290–2300.
37. Barua, D., Faeder, J. R., and Haugh, J. M. (2008) Computational models of tandem Src homology 2 domain interactions and application to phosphoinositide 3-kinase. *J. Biol. Chem.* 283, 7338–7345.
38. Mu, F. P., Williams, R. F., Unkefer, C. J., Unkefer, P. J., Faeder, J. R., and Hlavacek, W. S. (2007) Carbon-fate maps for metabolic reactions. *Bioinformatics* 23, 3193–3199.

39. Rubenstein, R., Gray, P. C., Cleland, T. J., Piltch, M. S., Hlavacek, W. S., Roberts, R. M., Ambrosiano, J., and Kim, J. I. (2007) Dynamics of the nucleated polymerization model of prion replication. *Biophys. Chem.* 125, 360–367.
40. Gillespie, D. T. (1976) A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. Comp. Phys.* 22, 403–434.
41. Gillespie, D. T. (1977) Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.* 81, 2340–2361.
42. Gillespie, D. T. (2007) Stochastic simulation of chemical kinetics. *Annu. Rev. Phys. Chem.* 58, 35–55.
43. Hucka, M., Finney, A., Sauro, H. M., Bolouri, H., Doyle, J. C., Kitano, H., Arkin, A. P., Bornstein, B. J., Bray, D., Cornish-Bowden, A., et al. (2003) The systems biology markup language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics* 19, 524–531.
44. Hoops, S., Sahle, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P., and Kummer, U. (2006) COPASI--a COmplex PAthway SImulator. *Bioinformatics* 22, 3067–3074.
45. Cohen, S. D., and Hindmarsh, A. C. (1996) CVODE, A Stiff/Nonstiff ODE Solver in C. *Comp. Phys.* 10, 138–143.
46. Hindmarsh, A. C., Brown, P. N., Grant, K. E., Lee, S. L., Serban, R., Shumaker, D. E., and Woodward, C. S. (2005) SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Softw.* 31, 363–96.
47. Berg, J. M., Tymoczko, J. L., and Stryer, L. (2006) *Biochemistry.* W. H. Freeman, New York.
48. Gross, J. L., and Yellen, J. (eds.) (2003) *Handbook of Graph Theory.*CRC Press, Boca Raton, FL.
49. McKay, B. D. (1981) Practical graph isomorphism. *Congressus Numerantium* 30, 45–87.
50. Ullmann, J. R. (1976) An algorithm for subgraph isomorphism. *J. ACM* 23, 31–42.
51. Lemons, N. and Hlavacek, W. S. private communication.
52. Borisov, N. M., Markevich, N. I., Hoek, J. B., and Kholodenko, B. N. (2005) Signaling through receptors and scaffolds: Independent interactions reduce combinatorial complexity. *Biophys. J.* 89, 951–966.
53. Borisov, N. M., Markevich, N. I., Hoek, J. B., and Kholodenko, B. N. (2006) Trading the micro-world of combinatorial complexity for the macro-world of protein interaction domains. *BioSystems* 83, 152–166.
54. Borisov, N. M., Chistopolsky, A. S., Kholodenko, B. N., and Faeder, J. R. (2008) Domain-oriented reduction of rule-based network models *IET Syst. Biol.* 2, 342–351.
55. Lauffenburger, D. A. and Linderman, J. J. (1993) *Receptors: Models for Binding, Trafficking, and Signalling.* Oxford, New York, NY.
56. Posner, R. G., Wofsy, C., and Goldstein, B. (1995) The kinetics of bivalent ligand-bivalent receptor aggregation: Ring formation and the breakdown of the equivalent site approximation. *Math. Biosci.* 126, 171–190.
57. Pollard, T. D. and Borisy, G. G. (2003) Cellular motility driven by assembly and disassembly of actin filaments. *Cell* 112, 453–465.
58. Koschorreck, M., Conzelmann, H., Ebert, S., Ederer, M., and Gilles, E. D. (2007) Reduced modeling of signal transduction – A modular approach. *BMC Bioinformatics* 8, 336.
59. Heath, J., Kwiatkowska, M., Norman, G., Parker, D., and Tymchyshyn, O. (2007) Probabilistic model checking of complex biological pathways. *Lect. Notes Comput. Sci.* 4210, 32–47.