

Rule-Based Modeling of Biological Systems using BioNetGen modeling language

Michael L. Blinov¹, James R. Faeder², William S. Hlavacek³

¹ Center for Cell Analysis and Modeling, University of Connecticut Health Center, Farmington, CT 06030, USA

² Department of Computational Biology, School of Medicine, University of Pittsburgh, Pittsburgh, PA 15260, USA

³ Theoretical Biology and Biophysics Group, Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM 87545, USA

E-mail: blinov@uchc.edu

1.	Introduction to Rule-based Modeling	2
1.1.	Modeling goals and challenges	2
1.2.	Conventional approach to model specification	2
2.	BNGL in a nutshell	3
2.1.	Organization of BNGL file	3
2.2.	The simplest BNGL model: a single reaction	4
2.3.	BNGL use to track composition and binding of species	5
2.4.	Introducing rules that generate multiple reactions and new species	7
2.5.	Introducing multi-state molecules	9
2.6.	Introducing dimers	11
2.7.	Investigating different protein-protein interaction mechanisms	12
2.8.	Changing parameters, network pre-equilibration and simulation	14
2.9.	Fluorescent labeling of molecules	14
2.10.	Introducing potentially infinite chains	16
2.11.	simulate_ode();	18
2.12.	Customization for interactions with the Virtual Cell	19
2.13.	Rate laws	20
2.14.	Symmetric Reaction Rules	20
2.15.	simulate_ssa();	21
3.	Rule-based modeling of complex signaling systems	22
3.1.	Epidermal Growth Factor Receptor Signaling	22
3.2.	FceRI immunoreceptor signaling	23
3.3.	Actin filaments	24
	References	25

1. Introduction to Rule-based Modeling

1.1. Modeling goals and challenges

What do we expect from a model?

- A model should incorporate details about quantities that can be measured and perturbed in experiments, so that predictions are testable.
- Parameters of a model should be independent of system behavior (i.e., they should have a physical rather than phenomenological basis),
- Analysis of a model should provide insights and guide experimentation by illuminating the logical consequences of knowledge and assumptions about the mechanistic details of a system, such as the interacting proteins, binding sites, enzymes responsible for post-translational modifications, and sites of modification in a system.

Signal-transduction systems generally involve post-translational modifications (such as phosphorylation), protein-protein interactions and the assembly of heterogeneous protein complexes. A signaling protein generally contains multiple binding sites as well as multiple sites subject to post-translational modifications (Yang, 2005). This multiplicity of sites gives rise to combinatorial complexity: the number of possible combinations of protein modifications and proteins in complexes grows exponentially with the number of functional sites in a system, and hundreds to thousands of chemical species may be generated by the interactions among only a few proteins (Morton-Firth, 1998; Kohn, 1999; Endy and Brent, 2001; Goldstein et al., 2002).

Consider, for example, the receptor tyrosine kinase EGFR. At least nine tyrosines in EGFR are phosphorylated during signaling (Jorissen, 2003). There are $2^9=512$ different phosphoforms of EGFR and $512*513/2= 131,328$ distinct combinations of these phosphoforms for a dimer of EGFR. The actual number of phosphorylation states of EGFR relevant for signaling in particular contexts may be much lower. However, to investigate the dynamics of phosphorylation of individual residues without bias, a modeler may want to consider the whole spectrum of possible phosphorylation states.

1.2. Conventional approach to model specification

The conventional approach to modeling a biochemical system is to draw a diagram depicting the chemical species and reactions in the system and then translate this scheme manually into a set of equations, such as a system of coupled ordinary differential equations (ODEs). A reaction scheme is an organized layout of a list of reactions that is based on a modeler's knowledge and assumptions about the system. Schemes for signal-transduction systems, which can be quite large (Oda et al., 2005), are based mostly on knowledge and assumptions about protein-protein interactions. A drawback of a reaction scheme is that it obscures the underlying protein-protein interactions, which are not explicitly represented. Still, a scheme is far easier to interpret than the corresponding equations, and the readability of a reaction scheme can be improved by using graphical

annotation designed with protein-protein interactions in mind (Kitano et al., 2005). In some cases, a reaction scheme can serve the purpose of model specification well.

Although the conventional approach to model specification is problematic for signal-transduction systems, which are marked by combinatorial complexity, it is nevertheless the approach most often used to specify models for these systems, but at a cost. Models derived in this way are invariably based on assumptions, which may be difficult to justify, that limit the chemical species and reactions considered to a fraction of those possible. An example is the model of Kholodenko et al. (1999) for EGFR signaling, which has been extended by a number of researchers (Schoeberl et al., 2002; Resat et al., 2003; Hatakeyama et al., 2003; Maly et al., 2004). This model is based on mechanistic assumptions that result in a selective focus on only a fraction of the protein complexes and phosphorylation states that could potentially arise from the protein-protein interactions considered in the model. For example, one assumption is that ligand-induced dimers of EGFR are unable to dissociate when receptors are phosphorylated, which seems unlikely. This assumption arises from a description of signaling events as an ordered pathway, which is consistent with the way this system is presented in typical diagrammatic interaction maps but inconsistent with rapidly reversible reactions and multiple branching possibilities. Lifting this and other assumptions causes a combinatorial explosion in the number of possible reactions and species (Blinov et al., 2006b), which makes manual model specification impractical.

2. BNGL in a nutshell

2.1. Organization of BNGL file

The model specification consists of four blocks, each beginning with a line containing `begin <blockname>` and ending with a line containing `end <blockname>`. Block names are

```
parameters
molecules (optional)
species or seed species
reaction rules
observables
```

They may appear in any order, although, because of the dependencies the above order is the most logical. The model specification is followed by a set of commands that operate on the model. Basic commands are

```
generate_network();
writeSBML();
simulate_ode({t_end=>NUMBER,n_steps=>NUMBER});
```

2.2. *The simplest BNGL model: a single reaction*

Consider the example of a single reaction of two species R and L forming a complex RL, $R + L \rightarrow RL$, with initial conditions $R(0)=R_0$, $L(0)=L_0$. In BNGL, we write it as

```
begin parameters
  R0  100
  L0  500
  RL0  0
  kon  0.01
  koff  0.1
end parameters

begin species
  R  R0
  L  L0
  RL  RL0
end species

begin reaction rules
  R + L <-> RL kon, koff
end reaction rules

begin observables
  Molecules R_unbound  R
  Molecules L_unbound  L
  Molecules R_complex_L  RL
  Molecules R_total  R RL
  Molecules L_total  L RL
end observables

generate_network();
writeSBML();
simulate_ode({t_end=>50,n_steps=>20});
```

Let us comment on some notations in the model:

- **Parameters** include initial concentrations and kinetic rate constants and are introduced without any units.
- **Reaction rules** specify a reversible mass-action reaction with parameters **kon** and **koff**.
- **Observables** define sums over the concentrations of species, which correspond to the quantities that are measured in typical biological experiments, such as total amount of unbound ligands (**L_unbound**) and ligands attached to receptors (**RL**).
- The first column of an observable declaration indicates the type of **observable (molecules)**, the next column is the name of the observable (it is used for displaying the results), and the remaining entries are species (separated by spaces) contributing to the observable.
- **Molecules** indicate a weighted sum over the species. In this example it is just a sum of concentrations.

- The `generate_network()`; command directs BioNetGen to generate a complete reaction network, which consists in this case of 3 species **R**, **L**, **RL** and 1 reversible reaction.
- The `create_sbml()`; command directs BioNetGen to save the reaction network in an SBML file.
- `simulate_ode({t_end=>50,n_steps=>20})`; directs BioNetGen to run a timecourse of 50 timeunits (consistent with your units used in parameters section) and report concentrations of all species and observables at 20 evenly-distributed time points.
- Please note that result of simulation **R_total** and **L_total** should be conserved over the time course.

2.3. *BNGL use to track composition and binding of species*

The example above can be modified to reflect the fact that RL is a complex of R and L. For that, we have to introduce a binding site on R for L: R(l), and a binding site on L for R: L(r). Names of binding sites are arbitrary, but just for convenience we denote binding site by the same letter as its binding partner. The complex RL will be represented directly as R bound to L through a chemical bond between their respective binding sites: R(l!1).L(r!1). Here the dot '.' denotes association of molecules into a complex, and '1' is the identifier of the chemical bond between r and l. Modifications to the old file are shown in red.

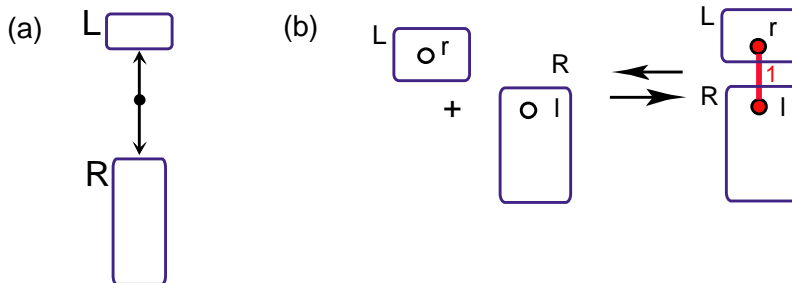


Figure 1 Model of simple ligand-receptor interaction. (a) Kohn Molecular Interaction map (MIM). (b) BioNetGen rule.

```
begin parameters
  R0  100
  L0  500
  kon  0.01
  koff 0.1
end parameters

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

```

end seed species

begin reaction rules
  R(1) + L(r) <-> R(1!1).L(r!1) kon, koff
end reaction rules

begin observables
  Molecules R_total R()
  Molecules L_total L()
  Molecules L_unbound L(r)
  Molecules R_unbound R(1)
  Molecules L_bound L(r!+)
  Molecules R_bound R(1!+)
  Molecules R_complex_L R().L()
  Molecules R_complex_L R(1!1).L(r!1)
end observables

generate_network();
writeSBML();
simulate_ode({t_end=>50,n_steps=>20});

```

Let us comment on some notations in the model:

- Binding sites introduced inside **Species** block.
- Instead of defining a separate species for the complex, the complex is expressed based on its components in **reaction rules** block.
- Entries in the **observable** block can be patterns that may select multiple species, for example:
 - **R()** selects all species containing a receptor, i.e. **R(1)** and **R(1!1).L(r!1)**
 - **L(r)** and **R(1)** indicate unbound forms of L and R respectively.
 - **R(1!+)** and **L(r!+)** indicate unbound forms of L and R respectively.
 - **R().L()** and **R(1!1).L(r!1)** indicate a complex of L and R
 - **R().L()**, **R(1!1).L(r!1)**, **R(1!+)**, **L(r!+)** in this particular example select the single species **R(1!1).L(r!1)**. However, as we'll see later, this will change as the model will be extended.

The reaction network generated by BioNetGen looks as follows:

```

begin species
  1 R(1)          R0
  2 L(r)          L0
  3 L(r!1).R(1!1) 0
end species
begin reactions
  1 1,2 3 kon
  2 3 1,2 koff
end reactions
begin groups
  1 R_total      1,3
  2 L_total      2,3
  3 L_unbound    2
  4 R_unbound    1
  5 L_bound      3
  6 R_bound      3

```

```

7 RL          3
8 RL          3
end groups
    
```

In **reactions** block and **group** block numbers refer to **species** numbers. In **reactions** block reactants are separated from products by a space, and individual reactants and products are separated by comma, so in the expanded form the reactions block should look like

```

1 L(r) + R(l) -> L(r!1).R(l!1) kon
2 L(r!1).R(l!1) -> L(r) + R(l) koff
    
```

The block **groups** include species that contribute to the observable. Thus, **R_total** includes two species **R(l)** and **L(r!1).R(l!1)**, **Ltotal** includes two species **L(r)** and **L(r!1).R(l!1)**, etc.

2.4. Introducing rules that generate multiple reactions and new species

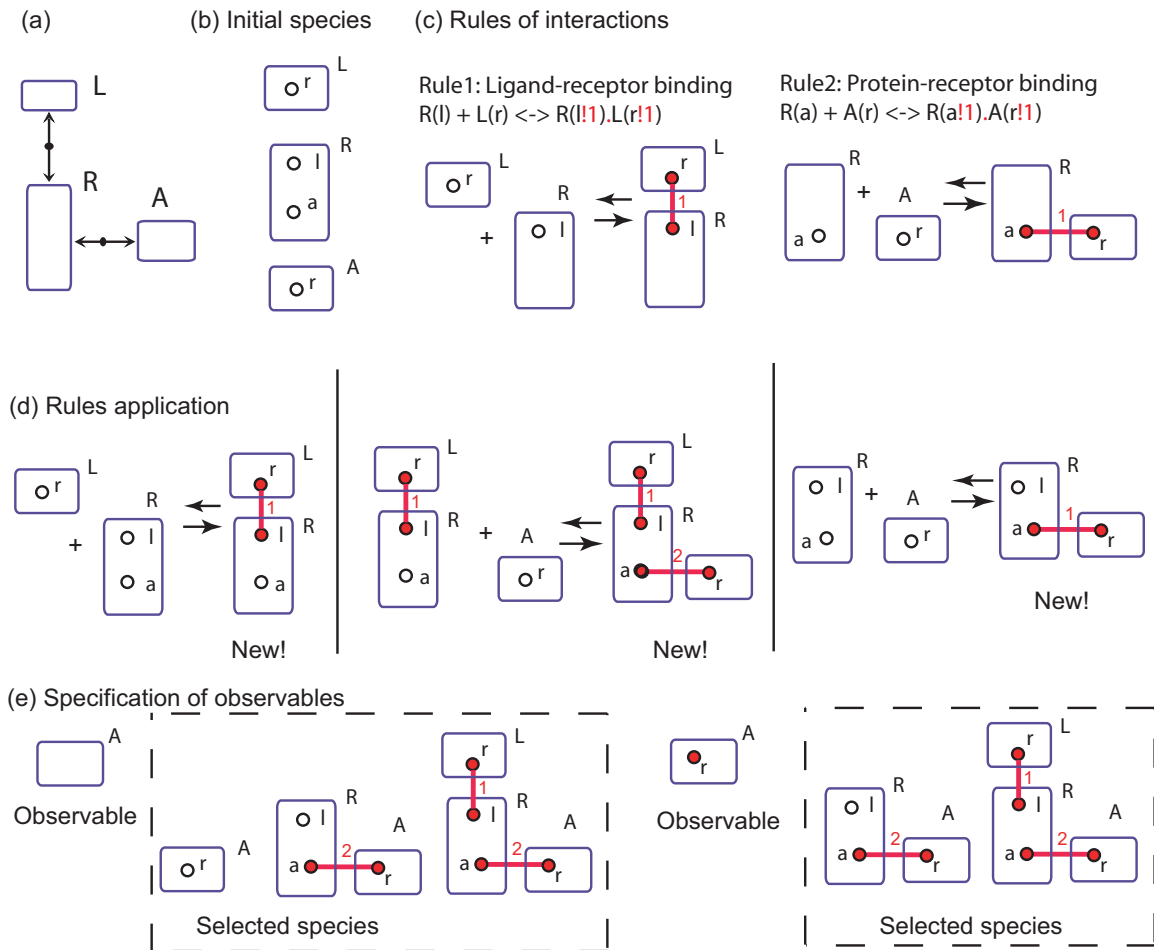


Figure 2 Model of independent ligand-receptor and protein-receptor interactions interaction.

Let us assume that protein A can bind R independently of L. The example above can be modified to include this feature. For that, we have to introduce:

1. An additional binding site on R for A: **R(l,a)**,
2. A new protein A with a binding site for R: **A(r)**.
3. New kinetic parameters: initial value of A **A0** and rates for binding of A to R: **kAon** and **kAoff**.
4. A reaction rule that specifies that A binds to R independently on the presence of a ligand L.

Modifications to the old file are shown in red.

```
begin parameters
  R0  100
  L0  500
  A0  100
  kon  0.01
  koff  0.1
  kAon 0.01
  kAoff 0.1
end parameters

begin seed species
  R(l,a)  R0
  L(r)    L0
  A(r)    A0
end seed species

begin reaction rules
  R(l) + L(r) <-> R(l!1).L(r!1) kon, koff
  R(a) + A(r) <-> R(a!1).A(r!1) kAon, kAoff
end reaction rules

begin observables
  Molecules Rtotal  R()
  Molecules Ltotal  L()
  Molecules Atotal  A()
  Molecules L_unbound  L(r)
  Molecules R_unbound  R(l,a)
  Molecules A_unbound  A(r)
  Molecules L_bound  L(r!+)
  Molecules R_bound  R(l!+)
  Molecules RLA  R().L().A()
  Molecules RA1  A(r!+)
  Molecules RA2  A(r!1).R(a!1)
  Molecules RA3  A().R()
end observables

generate_network();
writeSBML();
simulate_ode({t_end=>50,n_steps=>20});
```

The full model derived from these specifications consists of 6 species and 4 bidirectional reactions, while the model specification itself includes only 3 species and 2 rules.

To generate the model, we start from 3 species $S1=\mathbf{R(l,a)}$, $S2=\mathbf{L(r)}$, $S3=\mathbf{A(r)}$. Application of rule (1) to $S1$ and $S2$ generates a new species $S4=\mathbf{R(!l,a).L(r!1)}$. Application of rule (2) to $S1$ and $S3$ generates a species $S5=\mathbf{R(l,a!1).A(r!1)}$. Now the rule (1) can be applied to the species $S5$ to generate a new species $S6=\mathbf{R(!l,a!2).L(r!1).A(r!2)}$. Similarly, the rule (2) can be applied to the species $S5$ to generate the same new species $S6$.

Let us comment on the model:

- $\mathbf{R()}$ selects all species containing a receptor, 4 total; $\mathbf{A()}$ selects all species that contain A, 2 total; $\mathbf{L()}$ selects 3 species. They should be conserved during time course simulation.
- $\mathbf{L(r)}$ and $\mathbf{R(l,a)}$ indicate unbound forms of L and R respectively, thus select a single species each.
- If we would not change observable $\mathbf{R_unbound}$ from the last model and leave it as $\mathbf{R(I)}$, it would select only species unbound to a ligand, but potentially bound to A, thus it would be different from what we intend.
- Observables $\mathbf{RA1}$, $\mathbf{RA2}$ and $\mathbf{RA3}$ would select the same two species, a complex of A and R with and without ligand bound.

2.5. Introducing multi-state molecules

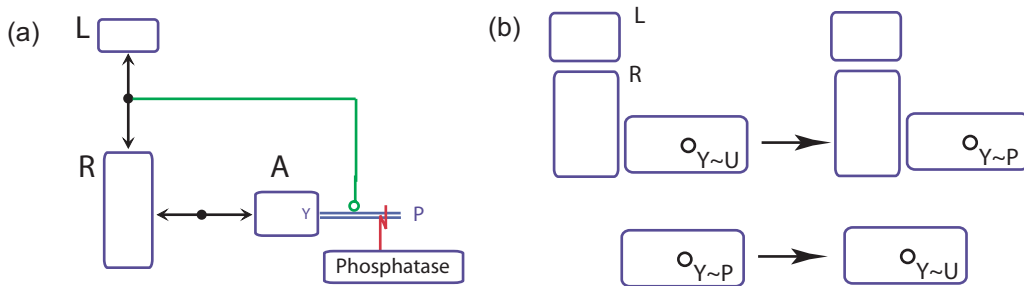


Figure 3 Model of ligand-induced protein phosphorylation. (a) Kohn Molecular Interaction map (MIM). (b) BioNetGen rule for phosphorylation/dephosphorylation.

Let us assume that R is the receptor tyrosine kinase, and A has a tyrosine subject to phosphorylation when A is bound to R. The example above can be modified to include this feature. For that, we have to introduce:

1. an additional phosphosite on A for A: $\mathbf{A(r,Y)}$
2. Two potential states of Y: phosphorylated (P) and unphosphorylated (U). This is introduced as $\mathbf{A(r,Y~U~P)}$. This declaration is optional.
3. Initial state of A $\mathbf{A(r,Y~U)}$ and additional rate constants \mathbf{kAp} and \mathbf{kAdp} .
4. A rule specifying that Y can be phosphorylated only when A is bound to R-L complex.
5. A rule specifying that A can be dephosphorylated at any time.

Modifications to the old file are shown in red.

```

begin parameters
  R0  100
  L0  500
  A0  100
  kon 0.01
  koff 0.1
  kAon 0.01
  kAoff 0.1
  kAp 0.01
  kAdp 0.1
end parameters

begin molecules
  R(l,a)
  L(r)
  A(r,Y~U~P)
end molecules

begin seed species
  R(l,a)  R0
  L(r)    L0
  A(r,Y~U)  A0
end seed species

begin reaction rules
  R(l) + L(r) <-> R(l!1).L(r!1) kon,koff
  R(a) + A(r) <-> R(a!1).A(r!1) kAon,kAoff
  L().R().A(Y~U) -> L().R().A(Y~P)  kAp
  A(Y~P) -> A(Y~U)  kAdp
end reaction rules

begin observables
  Molecules A_P  A(Y~P)
  Molecules A_unbound_P  A(r,Y~P)
  Molecules A_bound_P  A(r!+,Y~P)
  Molecules RLA_P  R().L().A(Y~P)
end observables

generate_network();
writeSBML();
simulate_ode({t_end=>50,n_steps=>20});

```

The full model consists of 9 species, 18 reactions (7 bidirectional reactions and 4 unidirectional reactions), although there are only 3 species and 4 rules in the model specification.

Species generated by the BioNetGen are $L(r!1).R(a,!1)$, $A(Y~U,r!1).R(a!1,l)$, $A(Y~U,r!1).L(r!2).R(a!1,!2)$, $A(Y~P,r!1).R(a!1,l)$, $A(Y~P,r!1).L(r!2).R(a!1,!2)$, $A(Y~P,r)$.

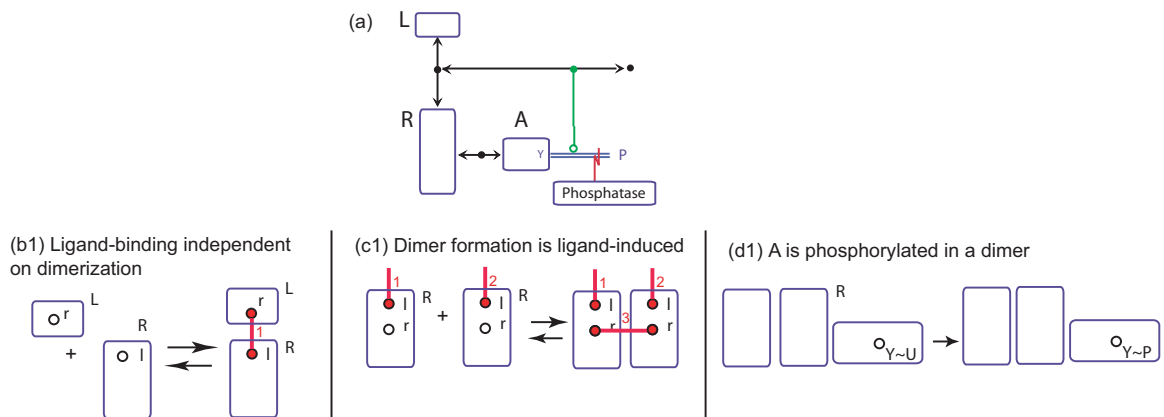
Let us comment on the model:

- A molecule block is an optional feature. If it is specified, it restricts potential states of Y to U and P. If this block is omitted, it will be generated by BioNetGen using reaction rules that introduce a new state P.
- Observable **A_P** contains the union of species of observable **A_unbound_P** and **A_bound_P**. It can be used as a check either in the network file, or as during the timecourse.

2.6. Introducing dimers

Let us now assume that R must be ligand-induced dimerized in order to transphosphorylate A. The example above can be modified to include this feature. For that, we have to introduce:

1. an additional receptor-binding site on R: R(l,r,a).
2. A rule for ligand-induced binding: two ligand-bound receptors can dimerize, independently on A.
3. Additional rate constants.
4. Modification to phosphorylation rule, stating that A can be phosphorylated only in a receptor-dimer.



Modifications to the old file are shown in red.

```

begin parameters
  R0 100
  L0 500
  A0 100
  kon 0.01
  koff 0.1
  kAon 0.01
  kAoff 0.1
  kAp 0.01
  kAdp 0.1
end parameters

begin molecules
  R(l,r,a)
  L(r)
  
```

```

    A(r,Y~U~P)
end molecules

begin seed species
  R(l,r,a)  R0
  L(r)     L0
  A(r,Y~U)  A0
end seed species

begin reaction rules
  R(l) + L(r) <-> R(l!1).L(r!1)    kon,koff
  R(a) + A(r) <-> R(a!1).A(r!1)    kAon,kAoff
  R(l!1,r) + R(l!2,r) <-> R(l!1,r!3).R(l!2,r!3)  kon,koff
  R().R().A(Y~U) -> R().R().A(Y~P)  kAp
  A(Y~P) -> A(Y~U)    kAdp
end reaction rules

begin observables
  Molecules R_Dim_M1  R().R()
  Molecules R_Dim_M2  R(r!+)
  Species R_Dim_S  R(r!+)
end observables

generate_network();
writeSBML();
simulate_ode({t_end=>50,n_steps=>20});

```

Let us comment on the model:

- The model now consists of 30 species and 137 reactions, so manual specification is already difficult.
- In observables block we introduce a new type **Species**. **Molecules** is the more common type and indicates a weighted sum over the species selected by the pattern(s) defining a group, with the weight given by the number of matches found for each species. For example, a dimer containing two receptors R would have a weight of 2 in the observable. On the other hand, **Species** would count the dimer once.

2.7. Investigating different protein-protein interaction mechanisms

Please note the flexibility in the ligand-binding mechanism:

- The rule written above allows a ligand to bind reversibly to any receptor. Thus, a ligand may dissociate from a receptor in a dimer, and dimers with one or no ligand may exist.
- To prevent ligand from dissociation in a dimer, one has to modify the ligand-receptor binding rule as follows:

```

  R(l) + L(r) -> R(l!1).L(r!1)    kon
  R(l!1,r).L(r!1) -> R(l,r) + L(r)  koff

```

Note that the number of reactions decreases to 47, species to 15.

- To prevent ligand from interacting with a receptor in a dimer, modify the first ligand-receptor binding rule as follows:

```

  R(l,r) + L(r) <-> R(l!1,r).L(r!1)    kon

```

We get 44 interactions among 15 species.

Please note the flexibility in the dimerization mechanism:

- The rule as written above allows only ligand-bound receptors to dimerize and break a dimer.
- If we expect that ligand can dissociate from a receptor in a dimer, we may allow any dimer to break-up. It is done by modifying a rule:

$$R(l!1,r) + R(l!2,r) \rightarrow R(l!1,r!3).R(l!2,r!3) \quad \text{kon}$$

$$R(r!3).R(r!3) \rightarrow R(r) + R(r) \quad \text{koff}$$
- Finally, we can request that even one ligand prevents dimer from break-up.

$$R(l!1,r) + R(l!2,r) \rightarrow R(l!1,r!3).R(l!2,r!3) \quad \text{kon}$$

$$R(l,r!3).R(l,r!3) \rightarrow R(l,r) + R(l,r) \quad \text{koff}$$

Please note the flexibility in the mechanisms providing phosphorylation of A:

- The rule as written above requires only two receptors in a complex for phosphorylation of A. The model does not prevent a ligand from binding/dissociation from the receptor, thus any dimer (with 1 or 2 ligands or without ligands at all) can phosphorylate A.

$$L().L().R().R().A(Y\sim U) \rightarrow L().L().R().R().A(Y\sim P) \quad \text{kAp}$$
- To restrict phosphorylation to L-R-R-L complexes, introduce

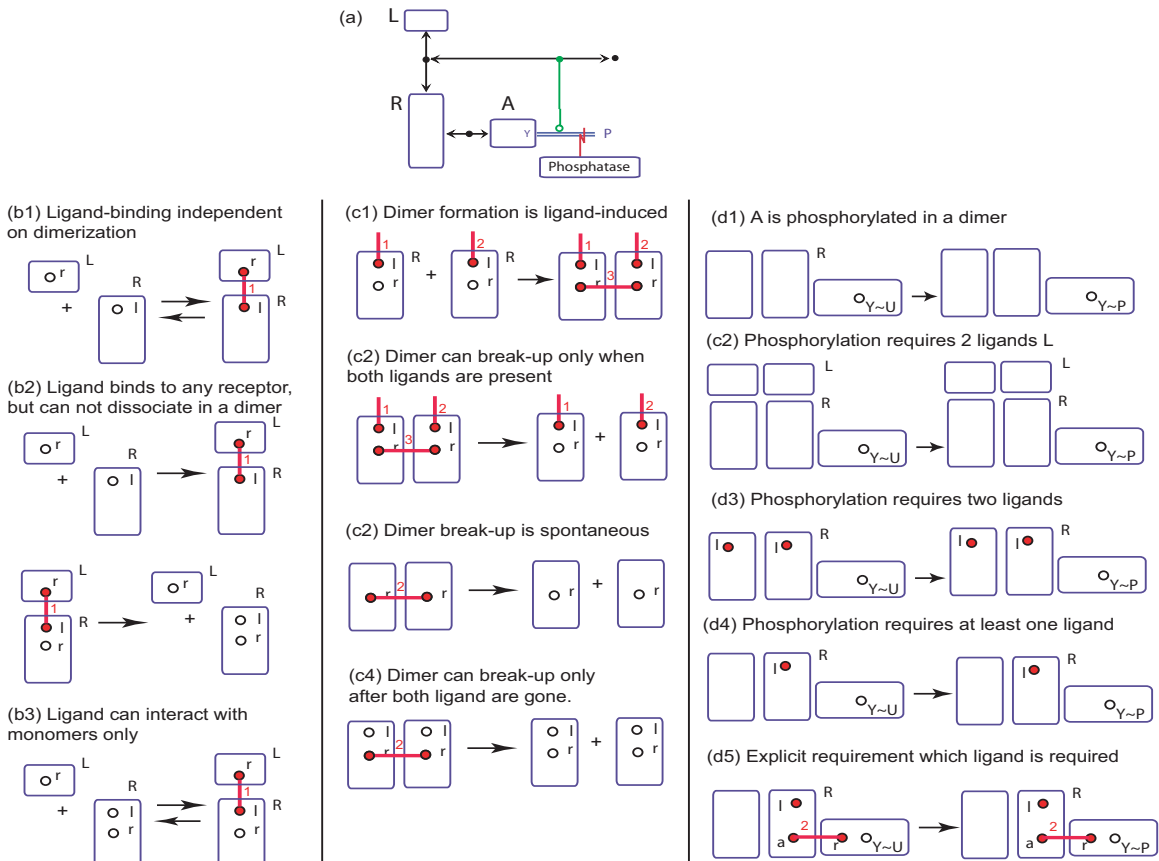
$$L().L().R().R().A(Y\sim U) \rightarrow L().L().R().R().A(Y\sim P) \quad \text{kAp}$$
- We may request that at least one ligand is required:

$$R().R(l!1).A(Y\sim U) \rightarrow R().R(l!1).A(Y\sim P) \quad \text{kAp}$$
- Finally, we can request which ligand is required by using explicit bonds:

$$L(r!1).R().R(l!1,a!2).A(r!2,Y\sim U) \rightarrow$$

$$L(r!1).R().R(l!1,a!2).A(r!2,Y\sim P) \quad \text{kAp}$$

Exercise: Find the number of species and reactions for each of the above cases.



2.8. Changing parameters, network pre-equilibration and simulation

Sometimes one needs to change parameters during simulation. This is done as follows:

```
generate_network({overwrite=>1});
writeSBML({suffix=>"initial"});
writeNET({suffix=>"initial"});

# Equilibration
setConcentration("L(r)",0);
setParameter("L0",200);
writeSBML({suffix=>"equil"});
writeNET({suffix=>"initial"});
simulate_ode({t_end=>100000,n_steps=>10,sparse=>1,steady_state=>1});

# Kinetics
setConcentration("L(r)","L0");
writeSBML({suffix=>"kinetics"});
writeNET({suffix=>"initial"});
simulate_ode({suffix=>"kinetics",t_end=>120,n_steps=>120,atol=>1e-8,rtol=>1e-8,sparse=>1});
```

Comments to the model:

- The `setParameter` command sets a parameter value.
- The `setConcentration` command sets the concentration of a particular species.
- `writeNET` and `writeSBML` output files when user requests. The requested suffix is added to the file name.
- Input and output files used by BioNetGen2.

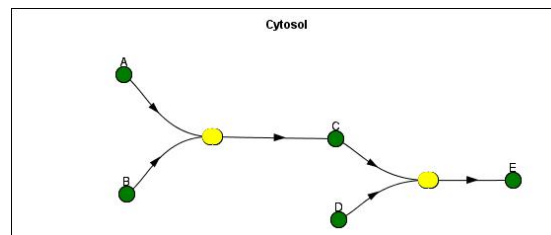
```
-----
|BNGL file|-> generate_network -> NET file (contains generated
|-----|                                     species, reactions, and observables)
-> simulate_{ode,ssa}-> CDAT file (concentrations of all species)
|-----|                                     -> GDAT file (concentrations of observables)
-> writeSBML -> XML file (contains parameters, species, reactions,
|-----|                                     and observables in SBML level 2format)
```

2.9. Fluorescent labeling of molecules

In some cases we have a reaction network and we add a property that is “carried through” the network. The simplest example is fluorescent labeling.

```
begin parameters
  A_fre    50
  B_tot    100
  D_tot    100
  E_tot    100
  k1f 1
  k1r 1
end parameters
```

```
begin species
```



```

A()  A_fre
B()  B_tot
C()  0
D()  D_tot
E()  0
end species

begin reaction rules
A() + B() <-> C() k1f, k1r
C() + D() <-> E() k1f, k1r
end reaction rules

generate_network({overwrite=>1});
writeSBML({});

```

In this example, we start from the simplest reaction network $A + B \rightarrow C$, $C + D \rightarrow E$. Now we fluorescently label A. Fluorescence is passed to C and E, and each fluorescent species can be bleached. The reaction network for fluorescent network is doubled in size: it has now 8 species and 7 reactions versus 5 species and 2 reactions of the initial reaction network. For larger networks, this expansion will be error-prone if done by hands. BioNetGen provides a mechanism for introducing the labels. We introduce:

- A label for each molecule that can be in fluorescent state: $A(\text{label}\sim\text{none}\sim\text{F})$, $B(\text{label}\sim\text{none}\sim\text{F})$, $C(\text{label}\sim\text{none}\sim\text{F})$.
- Extend reactions by introducing labels
- Introduce bleaching reactions.

```

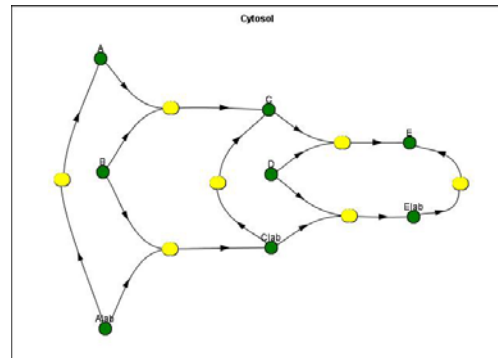
begin parameters
  A_fre      50
  A_flu      50
  B_tot      100
  D_tot      100
  E_tot      100
  k1f 1
  k1r 1
end parameters

begin molecule types
A(label~none~F)
B()
C(label~none~F)
D()
E(label~none~F)
end molecule types

begin species
A(label~none)  A_fre
A(label~F)     A_flu
B()            B_tot
C(label~none)  0
D()            D_tot
E(label~none)  0
end species

begin reaction rules

```



Rule-Based Modeling of Biological Systems using BioNetGen Modeling Language

```
A(label%1) + B() <-> C(label%1) k1f, k1r
C(label%1) + D() <-> E(label%1) k1f, k1r
A(label~F) -> A(label~none) k1f
C(label~F) -> C(label~none) k1f
E(label~F) -> E(label~none) k1f
end reaction rules
```

```
generate_network({overwrite=>1});
writeSBML({});
```

Another representation of the same process, even more compact:

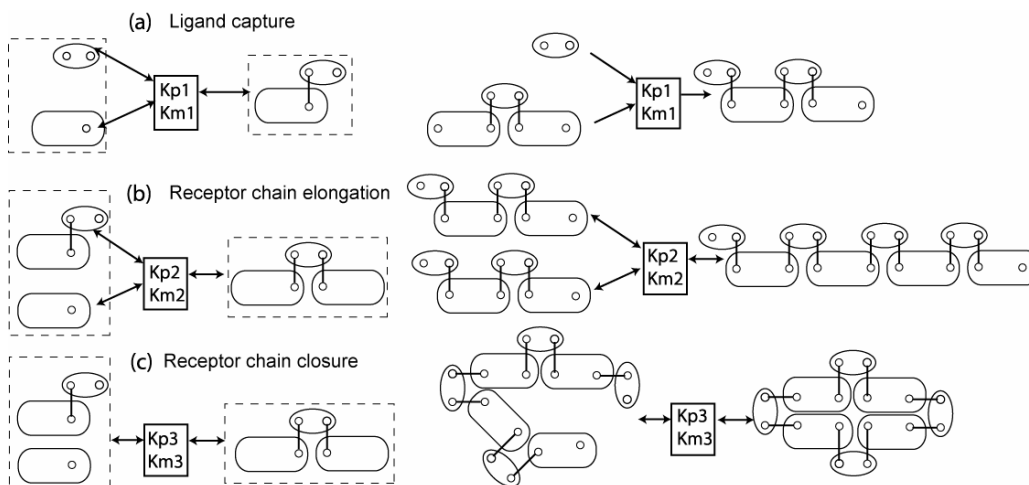
```
begin molecule types
A(label~none~F,in_complex~none~C~E)
B
D
end molecule types

begin species
A(label~none,in_complex~none) A_fre
A(label~F,in_complex~none) A_flu
B() B_tot
D() D_tot
end species

begin reaction rules
A(in_complex~none) + B <-> A(in_complex~C) k1f, k1r
A(in_complex~C) + D <-> A(in_complex~E) k1f, k1r
A(label~F) -> A(label~none) k1f
end reaction rules
```

2.10. Introducing potentially infinite chains

Let us now describe chains and rings formed by interactions of bivalent ligands and bivalent receptors. There are three rules: ligand binding to a receptor, chain elongation and ring closure. The model generated by BioNetGen includes all chains and rings consisting of no more than 5 ligand and 5 receptors - a total of 20 species and 92 interactions.




```

begin parameters
  kp1 0.00001
  km1 0.01
  kp2 0.00001
  km2 0.01
  kp3 0.00001
  km3 0.01
  R0 100
  L0 100
end parameters

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

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

generate_network({max_stoich=>{R=>5,L=>5}});

```

Let us comment on the model:

- The model now consists of 20 species and 92 reactions.
- You can add comments to the any place of the model. Everything on the line after # will be treated as a comment.
- The generate_network command generates a complete or partial network of species, reactions, and observables through iterative application of the rules to the initially defined species. For each iteration, the entire set of rules is applied to all of the current species, potentially generating new reactions and species. New species generated at the current iteration are not added to the species list until all of the rules have been applied. The order in which the rules are specified in the input file therefore does not affect the species and reactions generated at each iteration, although it will affect the order in which they appear in the species and reaction lists. The parameters that affect the behavior of the generate_network command are given in the Table below.

Table . Parameters for the generate_network command

Name	Function	Default value
check_iso	Perform isomorphism check for species that generate identical strings (keep this on unless you know what you're doing!)	1 (On)
max_agg	Max. number of molecules in one species	1e99
max_iter	Max. number of rule applications	100
max_stoich	Sets limit for number of molecules of each specified type in one species (hash- see syntax above)	Unset

Calling `generate_network` with the default parameters (no parameters specified) will cause network generation to proceed until the set of species and reactions no longer increases with further application of the rules. Some rules sets generate an infinite number of species and reactions, so in practice the maximum number of iterations is set to a finite value.

2.11. *simulate_ode()*;

The `simulate_ode` command is used to compute a timecourse of the network using ordinary differential equations to represent the average concentration of each species. The initial concentrations for species defined in the Species block are set to the declared values, while the concentrations of all species generated by `generate_network` are set to zero. `simulate_ode` calls the program Network, which provides an interface to the general-purpose ODE-solver CVODE. Adaptive implicit multi-step methods are used in the propagation to ensure efficient and accurate solution of the ODE's, which are usually stiff for networks with more than a few species. For very large systems (more than a few hundred species), then the `sparse=>1` option is recommended. (We find that most systems are sparse.) With this option, CVODE uses sparse matrix methods to avoid storage of the $n_{\text{species}} \times n_{\text{species}}$ Jacobian matrix, which becomes prohibitive for systems with more than about 10^4 species, and the iterative GMRES algorithm to solve the required systems of linear equations. This enables networks with 10^3 - 10^4 + species to be simulated in a few minutes (or less) on standard processors, even with stiffness.

Note that the parameter `sample_times`, which sets the times at which the concentrations are to be sampled, is an example of an array-valued parameter. Its argument is a comma-separated list of numbers enclosed by square brackets. The command

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

would cause the species concentrations and observable values to be printed at times of 0, 1, 10, and 100.

Table . Parameters for the `simulate_ode` command. * indicates required parameters.

Name	Function	Default value
<code>atol</code>	Absolute error tolerance for species concentrations	1e-8
<code>n_steps</code>	Number of intervals at which to report concentrations	1
<code>rtol</code>	Relative error tolerance for species concentrations	1e-8
<code>sample_times</code>	Times at which concentrations are reported (supercedes requirment for <code>t_end</code>)	none
<code>sparse</code>	Turns on use of sparse matrix formation of the Jacobian and iterative solution of linear equations using GMRES.	0 (Off)

	Recommended for networks with more than a few hundred species	
steady_state	Setting this to non-zero turns on check equilibration of species concentrations. Time course will stop when steady state is reached. If it is not reached in the allotted time the simulation will stop with an error.	0 (Off)
t_start	Starting time for integration	0
t_end*	End time for integration	None

2.12. Customization for interactions with the Virtual Cell

```

begin parameters
  A0 1000
  B0 500
  kp1 0.46e-5
  km1 1.0
  p 100
  d 5
  t 10
end parameters

begin species
  A(b,loc~Cyt) A0
  B(a,Y~U,loc~Cyt) B0
end species

begin reaction rules
  A(b,loc~Cyt) + B(a,loc~Cyt) <-> A(b!1,loc~Cyt).B(a!1,loc~Cyt) kp1,
  km1
  A(b!1).B(a!1,Y~U) -> A(b!1).B(a!1,Y~P) p
  B(Y~P,a,loc~Cyt) -> B(Y~P,a,loc~Nuc) t
  B(Y~P,loc~Nuc) -> B(Y~U,loc~Nuc) p
end reaction rules

begin observables
  Molecules A A()
  Molecules BP B(a!?,Y~P)
  Molecules BPC B(a!?,Y~P,loc~Cyt)
  Molecules BPN B(a!?,Y~P,loc~Nuc)
end observables

generate_network();
writeSBML();
simulate_ode({t_end=>50,n_steps=>20});

#%VC% mergeReversibleReactions
#%VC% speciesRenamePattern("\." , "_")
#%VC% speciesRenamePattern("[\(\,][a-zA-Z]\w*", "")
#%VC% speciesRenamePattern("~|\d*", "")
#%VC% speciesRenamePattern("\(\)", "")
#%VC% speciesRenamePattern("\)", "")
#%VC% setUnit("all", "default")

```

```

#%%VC% compartmentalizeSpecies("loc~Nuc", "Nucleus", "Cytoplasm")
#%%VC% compartmentalizeSpecies("loc~Cyt", "Cytoplasm", "")

```

2.13. Rate laws

Saturation and Michaelis-Menton rate laws can be invoked using one of the keywords for the allowed rate law types followed by a list of parameters in the parenthesis. The reaction

```
S + E -> P + E Sat(kcat,Km)
```

will have the rate law, $rate = kcat * [S] * [E] / (Km + [S])$. Note that the term in the denominator must appear first in the reaction rule.

Also, note that the second species (in this case 'E') is optional on the left hand side of the reaction, such that the reaction

```
E -> E + P Sat(kcat,Km)
```

will have the rate law, $rate = kcat * [E] / (Km + [E])$.

There is also a MM rate law type that gives the rate of the reaction corrected for the amount of S bound in the ES complex (solves quadratic formula for [S]free). This will give a closer approximation to the kinetics of the two elementary processes. The reaction

```
S + E -> P + E MM(kcat,Km)
```

will have the rate law, $rate = kcat * [S]_{free} * [E] / (Km + [S]_{free})$, where [S]_free is determined by

```
[S]_free = 0.5 * (([S] - Km - [E]) + (([S] - Km - [E])^2 + 4 * Km * [S])^(1/2))
```

2.14. Symmetric Reaction Rules

This is a confusing subject, so let's consider a simple example.

```
A(a) + A(a) -> A(a!1).A(a!1) kd
```

If A has a second domain b~U~P, then BNG will generate the following reactions

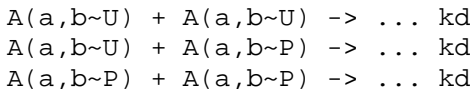
```

A(a,b~U) + A(a,b~U) -> ... 0.5*kd
A(a,b~U) + A(a,b~P) -> ... kd
A(a,b~P) + A(a,b~P) -> ... 0.5*kd

```

Only the second reaction agrees with what you think it should be. The extra factor of 0.5 is coming from the reaction kinetics, because the two reactants are the same, rather than

multiplicities. Consider two reactions $A + B \rightarrow \dots$ and $A + A \rightarrow \dots$. If $N_A = N_B$, the number of collisions per second for the first reaction is proportional to $N_A * N_B$ and for the second reaction is proportional to $N_A * (N_A - 1) / 2$. The factor of two that BNG is adding is due to the factor of two in the denominator. We decided to put this in the net file explicitly, because we assumed that most programs would not add the extra factor of two when computing the rates of symmetric reactions. If the factor of two is not added, then the total rate of reactions generated by the rule is not correct. In the example above, the rates of the reactions (without correction) would be



which neglects the fact that the symmetric reaction should occur at half the rate of the asymmetric one.

2.15. *simulate_ssa()*;

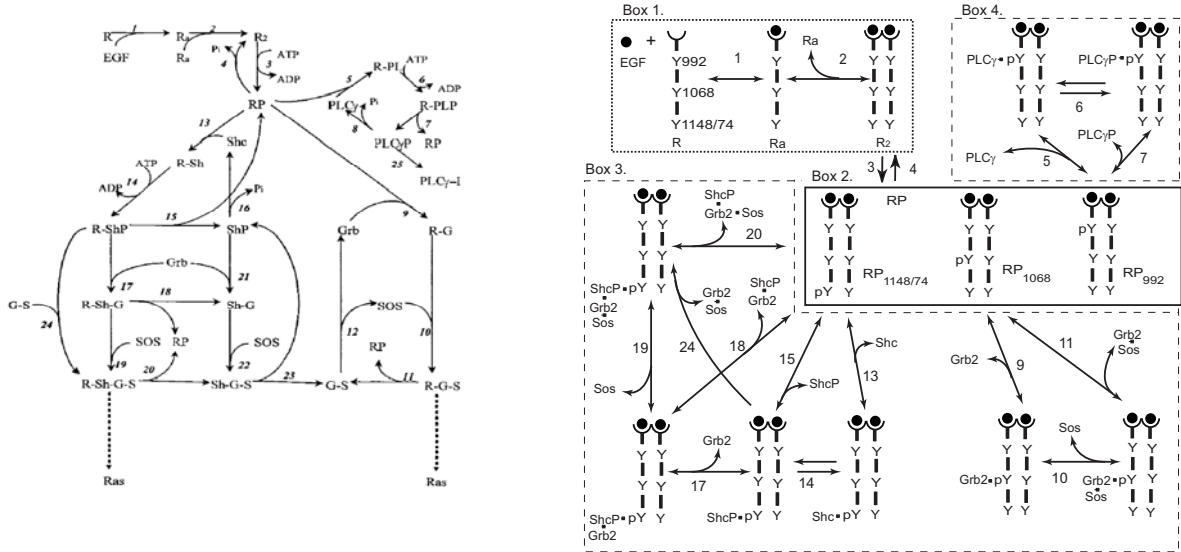
The `simulate_ssa` command is used to compute a timecourse of the network using the Gillespie direct algorithm for simulation of the stochastic master equations. When using this method to simulate a network, the species concentrations must be provided in number of molecules (per cell or per fraction of a cell) and the rate constants must be scaled accordingly. BNG2 does not currently provide any functions for performing unit conversions. The `simulate_ssa` interface is still incomplete and additional functionality will be added as needed. Currently, only one simulation run can be performed per command invocation.

The main difference between the `simulate_ssa` command and standard Gillespie implementations, is that species and reactions can be generated by application of the reaction rules on-the-fly. No special parameters are required to access this functionality. This is done by keeping track of whether the reaction rules have been applied to each species. When a species to which the rules have not been applied becomes populated for the first time, the rules are applied to that species to generate new reactions and species and to update observables. This adaptive generation of the network is particularly useful for infinite networks. A simulation can be carried out adaptively by setting a small value for `max_iter` in the `generate_network` command prior to the `simulate_ssa` command, say 1. The stochastic simulation will then map out the network over the course of the simulation.

Parameters for the `simulate_ssa` command are the same as for `simulate_ode`: `n_steps`, `sample_times`, `t_start`, `t_end`*

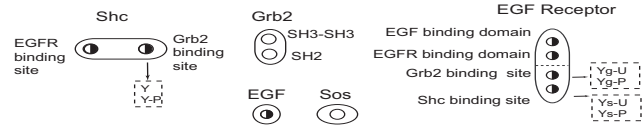
3. Rule-based modeling of complex signaling systems

3.1. Epidermal Growth Factor Receptor Signaling

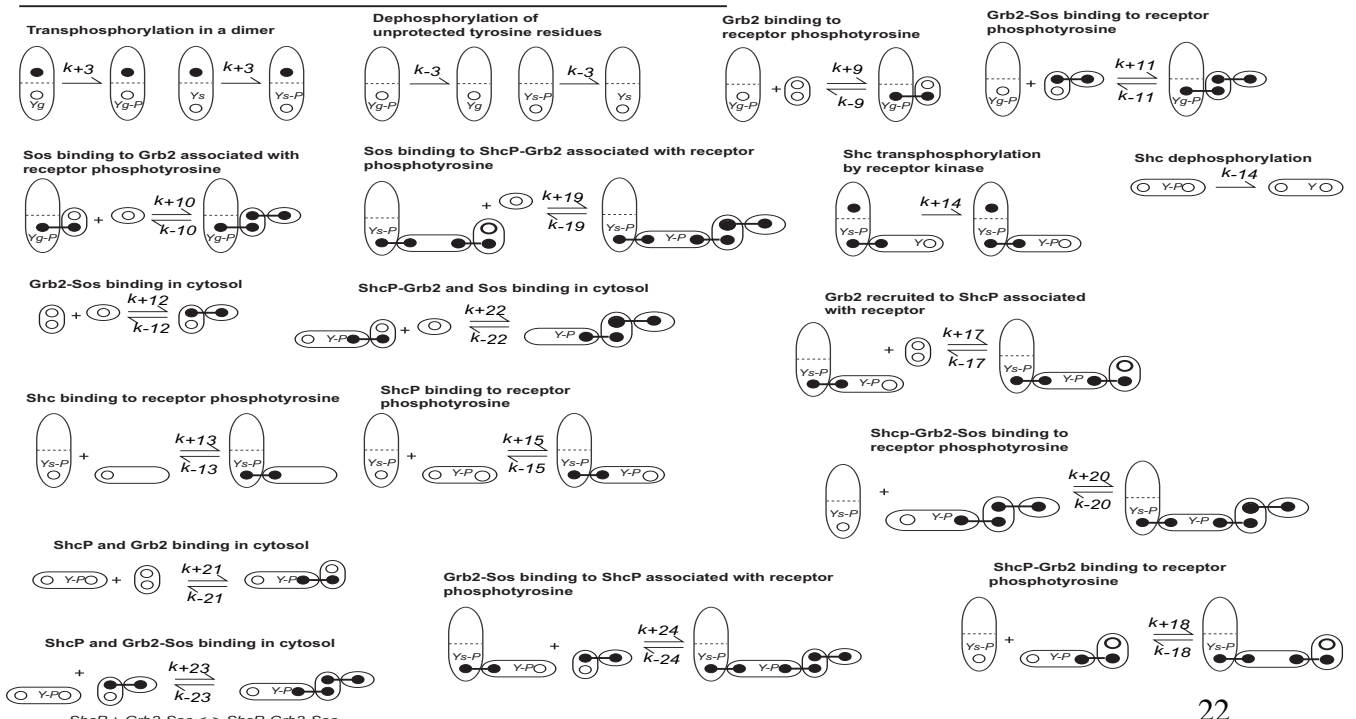
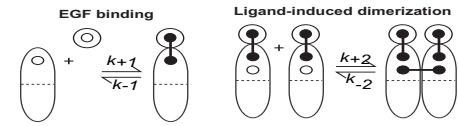


For discussion of results, see Blinov ML et al., BioSystems 2006.

(a) Molecules:



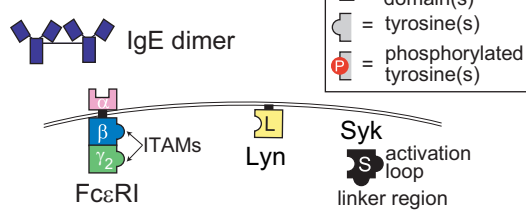
(B) Reaction rules



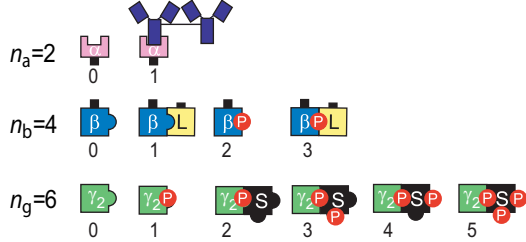
ShcP + Grb2-Sos \leftrightarrow ShcP-Grb2-Sos

3.2. *FcεRI* immunoreceptor signaling

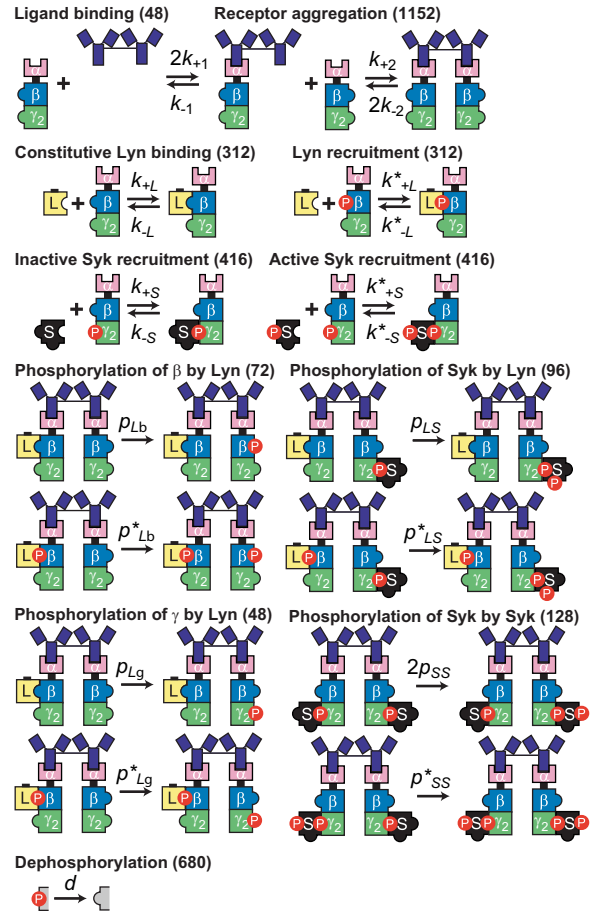
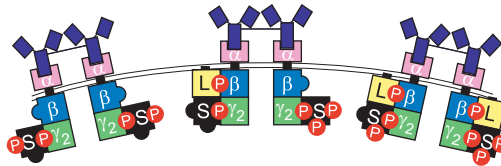
(a) Components



(b) Possible states of receptor subunits

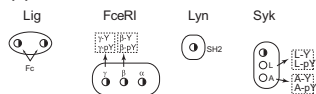


(c) A few of the 164 dimer states with active Syk

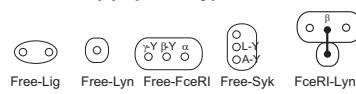


For discussion, see Faeder JR et al., JI 2003

(a) Molecules



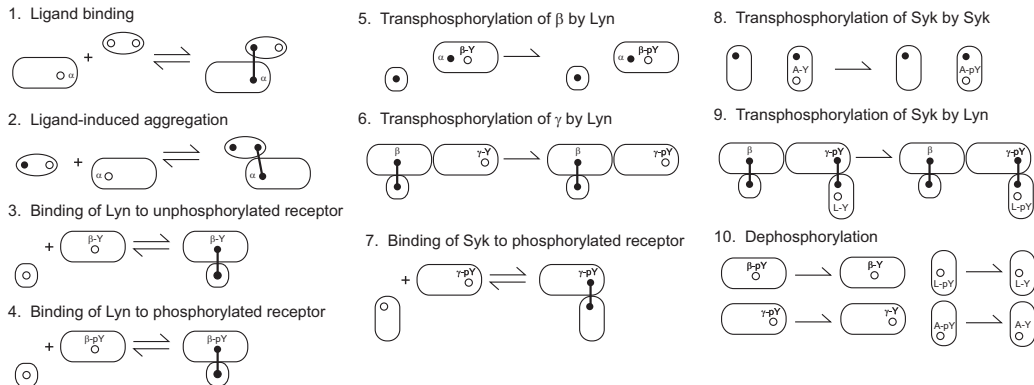
(b) Species types



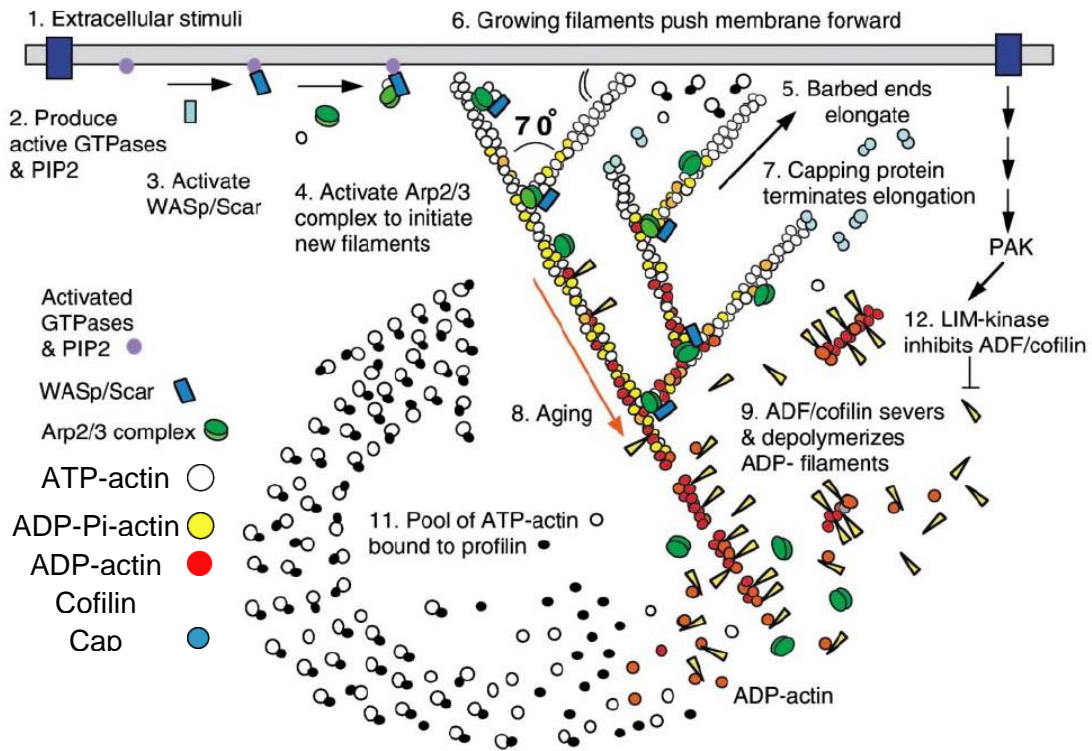
(c) Observables



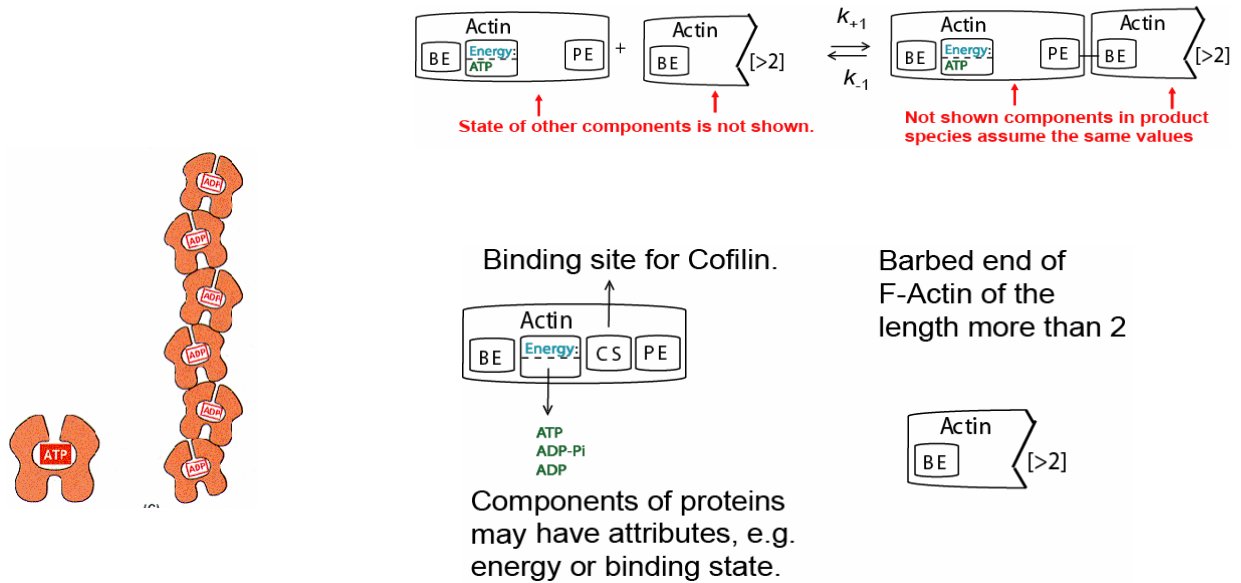
(d) Reaction Rules



3.3. Actin filaments



Pollard & Borisy, Cell 112 (2003)



References

Available at http://www.ccam.uchc.edu/mblinov/Blinov_publications.html

M. L. Blinov, J. Yang, J. R. Faeder and W. S. Hlavacek (2006) Graph theory for rule-based modeling of biochemical networks. *Transact. Computat. Syst. Biol. VII* in the series *Lect. Notes Comput. Sci.* **4230**, 89-106.

J. R. Faeder, M. L. Blinov, B. Goldstein and W. S. Hlavacek (2005) Rule-based modeling of biochemical networks. *Complexity* **10**, 22-41.

M. L. Blinov, J. R. Faeder, B. Goldstein and W. S. Hlavacek (2004) BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics* **20**, 3289-3291.

Contributors

BioNetGen, Los Alamos National Laboratory: Matthew Fricke, Jeremy Kozdon, Nathan Lemons, Ambarish Nag, Michael Monine, **Byron Goldstein**

BioNetGen@VCell, Center for Cell Analysis and Modeling, University of Connecticut Health Center: Mikhaill Levin, James Schaff, Anuradha Lakshminarayana, Fei Gao, Ion Moraru, **Leslie Loew**