Formal Approaches to Modelling Biomolecular Networks

Introduction to the Kappa modelling tool for rule-based modelling

Donal Stewart – 26-04-2012

Updated by David Sterratt – 23-04-2017

The material briefly covers the Kappa syntax and KaSim usage, then covers some common modelling techniques using some of Kappa's more useful features.

1 Installation

This tutorial is designed to run with KaSim version 4, which is still under development. To install on the following operating systems, get the appropriate "nightly build" binary from http://www.kappalanguage.org/nightly-builds/

- Windows: `KappaBin_master.zip`. Unpack this file. You then either need to set the Windows path to the directory contaning the KaSim.exe executable, or run KaSim from within R (see Section 3.1.3)
- Mac: `Kappapp.app.zip`
- Linux Ubuntu: `KaSa_master_x86_64_ubuntu16.04`

If you are not using one of these operating systems, consult the instructions at: https://github.com/Kappa-Dev/KaSim .

There are add-ons, such as a syntax highlighter for Windows Notepad++ editor and a Kappa mode for GNU Emacs, at http://dev.executableknowledge.org/ .

2 Source material and acknowledgements

The following sources were used to compile the material for this tutorial:

The KaSim manual at http://dev.executableknowledge.org/docs/KaSim-manual-master/KaSim_manual.htm
Vincent Danos' Synthetic Biology: Modelling MSc course at http://www.inf.ed.ac.uk/teaching/courses/sbm/

Thanks to Vincent Danos, Russ Harmer and Jean Krivine in particular.

3 Kappa syntax and KaSim use

This section covers the basics of the Kappa language, and the most commonly used parameters of the KaSim simulator.

3.1 Basic Kappa syntax & file structure

Tutorial file: simpleBinding.ka

```
# Simple binding example (the 'Hello World' of Kappa)

%agent: A(s)
%agent: B(s)

'Create bond' A(s), B(s) -> A(s!1), B(s!1) @ 1
'Break bond'  A(s!1), B(s!1) -> A(s), B(s) @ 200

%init: 1000 A(s)
%init: 1000 B(s)

%obs: 'Monomer A' |A(s)|
%obs: 'Dimer A-B' |A(s!1), B(s!1)|
```

Taking the components of the Kappa model individually

```
%agent: A(s)
%agent: B(s)
```

Agents are declared with the **`%agent:`** keyword, one agent per line. The agent declaration includes the agent name and all sites and states used by the agent in the Kappa model.

```
'Create bond' A(s), B(s) -> A(s!1), B(s!1) @ 1
'Break bond'  A(s!1), B(s!1) -> A(s), B(s) @ 200
```

Rules are the means by which a Kappa simulation evolves. They consist of an optional name such as **`'Create bond'`**, a transition **`A(s), B(s) -> A(s!1), B(s!1)`**, and a rate of transition **`@ 1`** .

The transition consists of a match pattern on the left, and an effect to apply on the right. In the first rule above, unlinked agent **`A`** and **`B`** are linked together by their respective sites **`s`**. In the second rule, the reverse process is written, with the link at the **`s`** sites being broken.

The rate clause will be discussed in more detail later.

```
%init: 1000 A()
%init: 1000 B()
```

The **`%init:`** clause allows you to state the initial composition of agents and complexes in the simulation. The lines above specify 1000 of agent **`A`** and 1000 of agent **`B`**, both with default states if any exist. Agent states will be discussed later.

```
%obs: 'Monomer A' |A(s)|
%obs: 'Dimer A-B' |A(s!1), B(s!1)|
```

Observations are recorded during the simulation of the model. The lines above specify patterns to match in the simulation state at each observation point. Each observation specification consists of a name **`'Monomer A'`** and a pattern to match, contained within a pair of vertical bars (pipe symbol) **`|A(s)|`** . The vertical bars, similar to the mathematical modulus operator, indicates the number of occurrences of a Kappa expression. More advanced observation specifications allow calculations to be performed using variables (described later) rather than a simple pattern match.

```
# Simple binding example (the 'Hello World' of Kappa)
```

Comments are any line suffix from the **`#`** to the end of the line.

3.1.1 Running the Kappa file in KaSim

KaSim is run from a command shell. Simulations can either be run by event or by time.

- By **event** means that the simulation ends after a specified number of events have passed, and that recording of observations happen after a regular number of events have occurred.
- By **time** means that the simulation ends after a specified amount of simulation time (i.e. time within the simulation, not elapsed time running the simulation) has passed, and that recording of observations happen regularly after a certain amount of simulation time has passed.

In the examples below

```
KaSim
```

means enter the operating system specific command line for the KaSim simulator.
Windows

```
KaSim
```

Linux

```
./KaSim
```

OSX

```
./KaSim
```

To run the Kappa model above by event for 1000 events

```
KaSim -i simpleBinding.ka -u event -l 1000
```

to run the model by time for 0.002 units (arbitrary but generally taken to be seconds)

```
KaSim -i simpleBinding.ka -u time -l 0.002 -p 0.00001
```

The output observations of the simulation by default are recorded in the file data.csv

```
8772
# "uuid" : "823756516"
"[T]", "'Monomer A'", "'Dimer A-B'"
0., 1000, 0
8.16681524079e-07, 999, 1
1.87412497934e-06, 998, 2
3.51428099129e-06, 997, 3
5.22039314531e-06, 996, 4
7.42092631802e-06, 995, 5
...
```

The first column records the simulation time at the observation point. Following this are the declared observations for the Kappa model, one per column, in this case recording the number of monomers and dimers.

In the first case above the “units”, as specified by the `-u event` argument, are events. The simulation will make 1000 observations (specified by the `-l` argument), evenly spread over the 1000 events.

In the second case the “units”, as specified by the `-u time` argument, are time. The observations will be spread over 2 ms (specified by the `-l` argument) and occur at intervals of 0.01ms (specified by the `-p` argument).

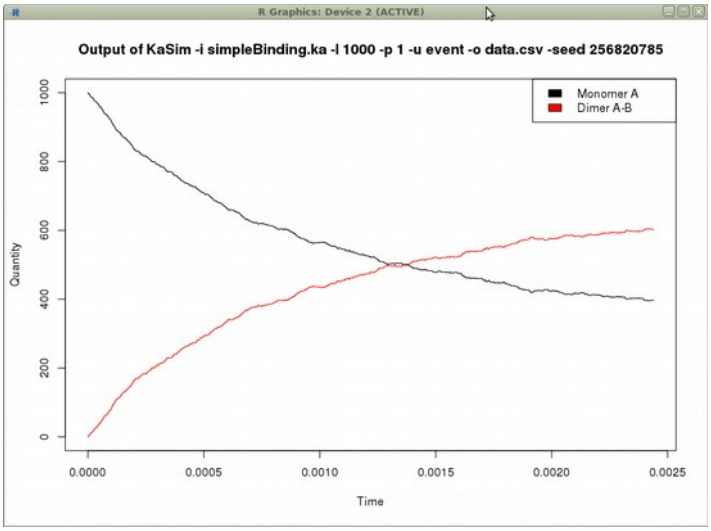
Try varying the granularity of the plot, by taking only 10 observations rather than 1000.

3.1.2 Visualising the KaSim output in R

Visualisation of the plot data can be done with whatever is your preferred means of plotting CSV data. For convenience you can also use the provided R file `kasim.R`. Install R from http://www.r-project.org . Open R and then type the following within the R terminal:

```
source(“kasim.R”)
plot.kappa()
```

A plot similar to the following should appear.



3.1.3 Running KaSim from R

It is also possible to run KaSim from within R. Edit the first line of `kasim.R` so it has the correct path to your KaSim executable file, which will look like this on Windows, if you have unpacked the `KappaBin_master.zip` file to "My Documents".

```
options(kasim="c:/users/my_user_name/Documents/KappaBin_master/KappaBin/bin/KaSim.exe")
```

Then you can run KaSim and capture its output using the run.kasim() command, and plot the output using plot.kasim():

```
source("kasim.R")
dat = run.kasim("simpleBinding.ka", l=1, p=0.01)
plot.kasim(dat)
```

Or for instant plots where you don't want to store the output:

```
plot.kasim(run.kasim("simpleBinding.ka", l=1, p=0.01))
```

3.2 Agent states, patterns & rule effects

Tutorial file: phosphorylation.ka

The example Kappa model describes the phosphorylation of a substrate by a kinase. Here a kinase may bind to an unphosphorylated substrate, and may either unbind with no effect, or unbind as the kinase phosphorylates the serine residue of the substrate.

```
# Phosphorylation of a substrate by a kinase

%agent: Substrate(s,serine~u~p)
%agent: Kinase(s)

'Kinase binding'   Substrate(s,serine~u), Kinase(s) \
                   -> Substrate(s!1,serine~u), Kinase(s!1) @ 1
'Kinase unbinding' Substrate(s!1), Kinase(s!1) \
                   -> Substrate(s), Kinase(s) @ 200
'Phosphorylation'  Substrate(s!1,serine~u), Kinase(s!1) \
                   -> Substrate(s,serine~p), Kinase(s) @ 200

%init: 1000 Substrate()
%init: 10 Kinase()
```

Note - The Kappa model above makes use of `\` to split long Kappa rules over multiple lines for clarity. As long as the `\` is the last character on the line, the Kappa declaration is taken to extend to the next line in the file.

Agents may contain multiple sites; agent `Substrate` above contains the sites `s` and `serine`. Each site may be used to link to other sites, and/or to maintain a state of the agent. In the example above, agent `Substrate` uses site `s` to link to the `Kinase`, and site `serine` to maintain phosphorylation state, `u` (unphosphorylated) or `p` (phosphorylated).

States are identified using the `~` symbol. All states of an agent site must be declared in the `%agent:` specification. The first state in this declaration is taken to be the default state of the site. When `%init:` declarations do not specify the state of the agent site, the default state is used.

Links are specified using the `!` symbol followed by an integer. There are always matched pairs of integers representing the start and end of a link between agent sites. In addition, when specifying links in rules, wildcards can be used: `site!_` means the site is linked, but unspecific as to what it is linked to; `site` means the site is not bound; `site?` means the site may or may not be bound. This can also be specified by leaving the site out of the rule entirely.

The example model above contains no observations. Add observations for the following to the Kappa model and run the simulations – are the results what you would expect ?

- **Unphosphorylated substrate**
- **Phosphorylated substrate**
- **Bound substrate (specifying link)**
- **Bound substrate (using link wildcard)**
- **Unbound substrate**
- **Both bound and unbound substrate**
- **Both bound and unbound substrate (using link wildcard)**

How would you modify the model above to allow binding of the kinase to the substrate regardless of the phosphorylation state of the substrate. Are the results as you would expect ?

So far, we have seen rules which change the state of an agent, and create and break links between agent sites. It is also possible to both create and destroy agents. The following Kappa rules extend the phosphorylation model to show gradual decomposition of the phosphorylated substrate into an intermediate compound, and its removal from the model.

```
'Breakdown'         Substrate(s,serine~p) -> BreakdownProduct() @ 1
'Removal'           BreakdownProduct() -> @ 1
```

Add these rules to the model, add observations to track the progress of the breakdown product, and modify the model to make it work (one extra line needed). Run the simulation and comment on the results.

Try constitutively adding unphosphorylated substrate to the model (i.e. making the substrate appear using a rule) at a rate of 100 per second. What are the effects ?

3.3 Rates

Rules use the suffix `@ VALUE` to specify the stochastic rate of that rule, i.e. the rate a rule is applied per instance of that rule. For example in the phosphorylation model above, the rule `'Kinase binding'` will occur at the rate of 1 for every combination of `Substrate(s,serine~u)` (initially 1000) and `Kinase(s)` (initially 10) present at that point in time during the simulation. For a complete discussion of stochastic rates and their relation to concentration based rates (k), please refer to section 3.4.4 (“Rates”) of the KaSim manual available at http://dev.executableknowledge.org/docs/KaSim-manual-master/KaSim_manual.htm

In the simplebinding.ka model earlier, why are the rates of the rules `'Create bond'` and `'Break bond'` of different orders of magnitude ? Hint, say there were 500 each of `A(s)`, `B(s)` and the bound dimer `A(s!1), B(s!1)`. What would the be the effective rates of the rules ?

3.4 Variables

Tutorial file: phosphorylationVariables.ka
Variables are declared using the `%var:` syntax. They can be used to symbolise either Kappa expressions or variable expressions involving calculations. Using the phosphorylation model above with variables added

```
# Phosphorylation of a substrate by a kinase

%agent: Substrate(s,serine~u~p)
%agent: Kinase(s)

'Kinase binding'   Substrate(s,serine~u), Kinase(s) \
                   -> Substrate(s!1,serine~u), Kinase(s!1) @ 'bind rate'
'Kinase unbinding' Substrate(s!1), Kinase(s!1) \
                   -> Substrate(s), Kinase(s) @ 'unbind rate'
'Phosphorylation'  Substrate(s!1,serine~u), Kinase(s!1) \
                   -> Substrate(s,serine~p), Kinase(s) @ 'phosp. rate'

%init: 'initial substrate' Substrate()
%init: 'initial kinase' Kinase()

%var: 'bind rate' 1
%var: 'unbind rate' 200
%var: 'phosp. rate' 'unbind rate'
%var: 'initial kinase' 10
%var: 'initial substrate' 'initial kinase' * 100

%var: 'bound substrate' Substrate(s!1), Kinase(s!1)
```

Variables consist of the declared variable name `'unbind rate'` followed by a definition of the value of the variable (in this case the value 200). Note the use of calculations and referencing other variables in defining variable values. The following are allowed symbols in variable definitions, taken from Table 4.3 of the KaSim manual.

Symbol	Interpretation
`[E]`	the number of simulation events since the beginning of the simulation
`[T]`	the bio-time of the simulation
`'v'`	the value of variable `'v'`
`[f]`	the intuitive mathematical function or constant associated to f ∈ {`log`,`sin`,`cos`,`tan`,`sqrt`,`pi`}
`[inf]`	symbol for infinity
`[mod]`	the modulo operator
`[exp]`	the exponentiation operation $x \rightarrow e^x$
`[int]`	the integer part $x \in R \rightarrow \lvert x \rvert \in N+$
`+,-,*,/,^`	the corresponding mathematical operators

3.5 Plot declaration

An alternative to using `%obs:` to declare an observable to be output in the output file is to declare a variable such as `'bound substrate'` in the section above, and then instruct the model to plot a named variable

```
%plot: 'bound substrate'
```

Experiment with variable options. Plot the variables and run KaSim on your models to check variable syntax and behaviour.

Create a variable `'Effective bind rate'` which holds the effective kinase binding rate in the model above (i.e. using the concentrations of the required agents of the rule). Plot the variable and check the simulation results are as expected.

4 Kappa techniques

With the basic Kappa syntax covered, we can continue with commonly used techniques in Kappa modelling.

4.1 Resolving ambiguous molecularity

In the phosphorylation example above, the rule `'Kinase binding'` could only bind unbound substrate to kinase

```
'Kinase binding'    Substrate(s,serine~u), Kinase(s) \
                    -> Substrate(s!1,serine~u), Kinase(s!1) @ 'bind rate'
```

If it was possible for the substrate to be already bound to the kinase through another site, for example

```
Substrate(s,serine~u,other!1), Other(sub!1,kinase!2), Kinase(s,other!2)
```

then the rule `'Kinase binding'` would have an inconsistent rate, depending on whether the substrate and kinase were bound or not. This is termed ambiguous molecularity, and should be avoided in Kappa models.

How would you change the Kappa model to separate the two applications of `'Kinase binding'` ?

In the cases where it is not easy to separate the ambiguous rules, Kappa allows you to specify two rates for a given rule. One to be used when the agents are unconnected, and the other when the agents are connected. For example

```
'Kinase binding'    Substrate(s,serine~u), Kinase(s) \
                    -> Substrate(s!1,serine~u), Kinase(s!1) \
                    @ 'unconnected bind rate' ('connected bind rate')
```

However, the rule above may cause the simulation to run much more slowly in KaSim. It is better to separate ambiguous rules where possible.

4.2 Use of scaling

In a stochastic system, the fewer instances of agents used, the noisier the system. For example

Take the simpleBinding model at the start of this session and modify the number of agents `A` and `B` to have only 100 instances each. Compare the resulting output from the simulation with that of the original model.

4.3 Multiple KaSim input files

KaSim can take multiple input files, and will simulate the model constructed by concatenating the input files together. This allows you to separate changeable parameters such as initial instance values, or rate variables into a separate file from the rest of the model. Then the Kappa model may be run using different parameters by simply using a different parameter file for each run

```
KaSim -i simpleBinding.ka -i parameters.ka -l 1000 -u event
```

4.4 Perturbations

The `%mod:` command allows the state of a simulation to be altered while the simulation is running. This can be used to introduce or remove reagents at certain time points, modify kinetic rates, and take snapshots of the mix of agents in the running simulation.

```
%mod: boolean_expression do effect
%mod: repeat boolean_expression do effect until boolean_expression
```

The first line above will run once on activation. The second line will run repeatedly on activation until the terminating condition is true. The boolean expressions can include waiting for a certain number of events to pass, an amount of time to have elapsed, the concentration of a group of agents in the simulation to reach a certain level, or any combination of these.

```
%mod: [T] > 1 do 'Kinase binding' := 2
```

Adding the above rule to the phosphorylation model will double the binding rate after 1 s has passed in the simulation.

The KaSim Manual section 6.1 shows the available perturbations.

Add a rule to the phosphorylation model to add another 1000 unphosphorylated substrate after 1.5s. You may need to increase the number of events run by the simulation.

Take a snapshot of the simulation after 1s. Open the created snapshot file in an editor. How could you use this file for future simulations ? Try it.

Formal Approaches to Modelling Biomolecular Networks
Application of rule-based modelling to signal transduction pathways
26-04-2012

This tutorial covers the use of Kappa in developing a well known signal transduction pathway, the mitogen-activated protein kinase (MAPK) cascade. We will incrementally develop the Kappa model, then experiment with the model structure and rates to determine the effects on the system being modelled.

Kappa models are available for the end result, but please attempt to create the components requested in the exercises yourselves first if possible.

1 Source material and acknowledgements

The following sources were used to compile the material for this tutorial:

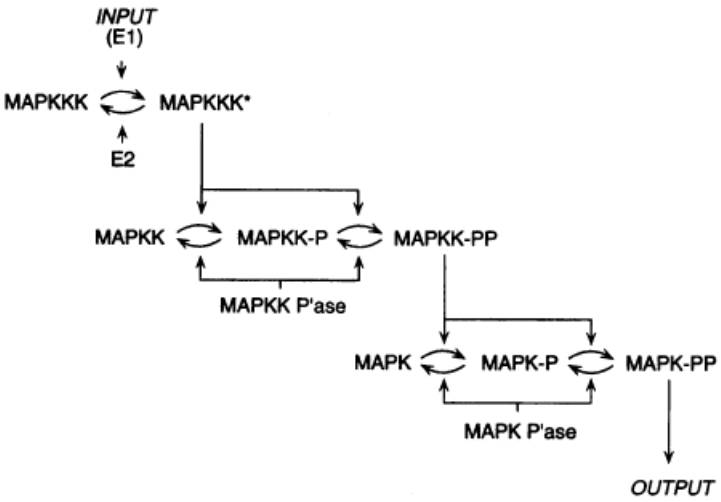
The original Huang & Ferrell paper 'Ultrasensitivity in the mitogen-activated protein kinase cascade' at http://www.pnas.org/content/93/19/10078.short
Figures below were taken from this paper.

MAPK by Russ Harmer (at http://www.rulebase.org/models/mapk). In particular, many of the exercises of this tutorial were taken from Russ' MAPK tutorial.
The KaSim manual at https://github.com/downloads/jkrivine/KaSim/KaSim_manual.pdf
A SBML version of the model at http://www.ebi.ac.uk/biomodels-main/BIOMD0000000009

Thanks to Vincent Danos, Russ Harmer and Jean Krivine in particular.

2 Biological background

The MAPK cascade is a signal and sensitivity amplification cascade know to operate in many biological pathways. An input signal (ligand) triggers the activation of a MAP kinase kinase kinase (MAPKKK). This activated MAPKKK causes double phosphorylation of its substrate MAP kinase kinase (MAPKK), which then in turn causes double phosphorylation of its substrate MAP kinase (MAPK). The cascade allows a small level of input signal to quickly trigger a large response, the activated MAPK.



Each one of the transitions in the figure above is a standard enzymatic reaction, just like the phosphorylation reaction covered in the earlier tutorial. So for example, the transition from MAPKKK to MAPKKK* above is

$$\text{MAPKKK} + \text{E1} \rightleftharpoons \text{MAPKKK-E1} \rightarrow \text{MAPKKK*} + \text{E1}$$

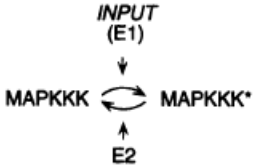
which in Kappa is represented with 3 rules.

3 Model construction

We will build the MAPK model incrementally. The first step will be to construct a model representing the first stage of the cascade.

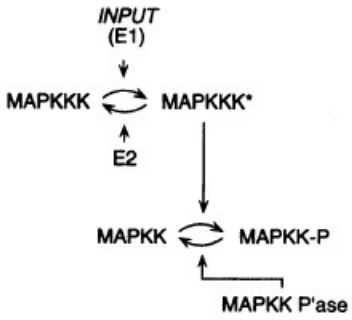
The model below represents the first step in the cascade.

Try simulating the model in KaSim and look at the output plots. Does it behave as expected ?

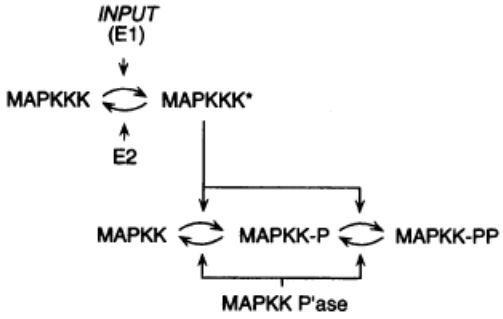



```
%agent: MAPKKK(e1,e2,pSite~u~p)
%agent: E1(kkk,state~active~inactive)
%agent: E2(kkk)

%var: 'rescale' 1.0

%var: 'BIND' 0.0001 / 'rescale'
%var: 'BREAK' 0.1
%var: 'MODIFY' 0.1

%init: 1000 E1
%init: 100 MAPKKK
%init: 10 E2

%obs: 'MAPKKKp' MAPKKK(pSite~p?)
%obs: 'E1p' E1(state~active?)


### MAPKKK ###

MAPKKK(e1,pSite~u), E1(kkk,state~active) \
    -> MAPKKK(e1!1, pSite~u), E1(kkk!1,state~active) \
    @ 'BIND' # binding of MAPKKK activator
MAPKKK(e1!1,pSite~u), E1(kkk!1) -> MAPKKK(e1, pSite~u), E1(kkk) \
    @ 'BREAK' # unbinding of MAPKKK activator
MAPKKK(e1!1,pSite~u), E1(kkk!1) -> MAPKKK(e1, pSite~p), E1(kkk) \
    @ 'MODIFY' # MAPKKK activation

MAPKKK(e2,pSite~p), E2(kkk) -> MAPKKK(e2!1, pSite~p), E2(kkk!1) \
    @ 'BIND' # binding of MAPKKK inactivator
MAPKKK(e2!1,pSite~p), E2(kkk!1) -> MAPKKK(e2, pSite~p), E2(kkk) \
    @ 'BREAK' # unbinding of MAPKKK inactivator
MAPKKK(e2!1,pSite~p), E2(kkk!1) -> MAPKKK(e2, pSite~u), E2(kkk) \
    @ 'MODIFY' # MAPKKK inactivation
```

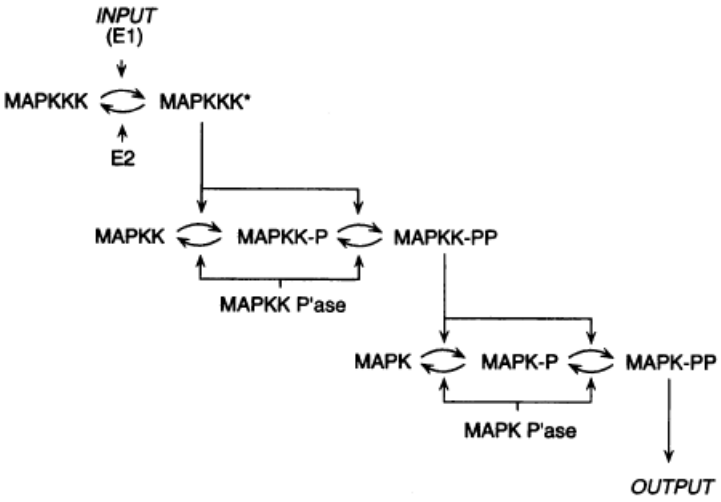
Now extend this model to include the first phosphorylation of the next step in the cascade. Use the same structure of bind/unbind/activate rules. Check simulation of the model still behaves well.



Add the second phosphorylation of this stage of the model, and again check the simulation. Allow the sites to be phosphorylated independently, i.e. separate phosphorylation sites in the agent `MAPKK`.



Finally, add the remaining stage of the cascade, and check the simulation results.



You should have ended up with a Kappa model roughly equivalent to the model below.
Tutorial file: mapk.ka

```
# Model of MAPK cascade
# from Ultrasensitivity in the mitogen-activated protein kinase cascade.
# Huang CY, Ferrell JE. 1996
#
# Model adapted from Kappa exercises by Russ Harmer
# http://www.rulebase.org/models/mapk


%agent: MAPKKK(e1,e2,kk,pSite~u~p)
%agent: MAPKK(k,pSite1~u~p,pSite2~u~p)
%agent: MAPK(pSite1~u~p,pSite2~u~p)
%agent: E1(kkk,state~active~inactive)
%agent: E2(kkk)
%agent: MAPKK-Pase(kk)
%agent: MAPK-Pase(k)

%var: 'rescale' 1.0

%var: 'BIND' 0.0001 / 'rescale'
%var: 'BREAK' 0.1
%var: 'MODIFY' 0.1

%init: 1000 E1
%init: 100 MAPKKK
%init: 500 MAPKK
%init: 1000 MAPK
%init: 10 E2
%init: 50 MAPKK-Pase
%init: 100 MAPK-Pase

%obs: 'MAPKpp' MAPK(pSite1~p?,pSite2~p?)
%obs: 'MAPKKpp' MAPKK(pSite1~p?,pSite2~p?)
%obs: 'MAPKKKp' MAPKKK(pSite~p?)
%obs: 'E1p' E1(state~active?)


### MAPKKK ###

MAPKKK(e1,pSite~u), E1(kkk,state~active) \
    -> MAPKKK(e1!1, pSite~u), E1(kkk!1,state~active) \
    @ 'BIND' # binding of MAPKKK activator
MAPKKK(e1!1,pSite~u), E1(kkk!1) -> MAPKKK(e1, pSite~u), E1(kkk) \
    @ 'BREAK' # unbinding of MAPKKK activator
MAPKKK(e1!1,pSite~u), E1(kkk!1) -> MAPKKK(e1, pSite~p), E1(kkk) \
    @ 'MODIFY' # MAPKKK activation

MAPKKK(e2,pSite~p), E2(kkk) -> MAPKKK(e2!1, pSite~p), E2(kkk!1) \
    @ 'BIND' # binding of MAPKKK inactivator
MAPKKK(e2!1,pSite~p), E2(kkk!1) -> MAPKKK(e2, pSite~p), E2(kkk) \
    @ 'BREAK' # unbinding of MAPKKK inactivator
MAPKKK(e2!1,pSite~p), E2(kkk!1) -> MAPKKK(e2, pSite~u), E2(kkk) \
    @ 'MODIFY' # MAPKKK inactivation

### MAPKK ###

MAPKK(pSite1~u), MAPKKK(pSite~p,kk) \
    -> MAPKK(pSite1~u!1), MAPKKK(pSite~p, kk!1) \
    @ 'BIND' # binding MAPKKK-P and MAPKK
MAPKK(pSite1~u!1), MAPKKK(kk!1) -> MAPKK(pSite1~u), MAPKKK(kk) \
    @ 'BREAK' # unbinding MAPKKK-P and MAPKK
MAPKK(pSite1~u!1), MAPKKK(kk!1) -> MAPKK(pSite1~p), MAPKKK(kk) \
    @ 'MODIFY' # phosphorylation of MAPKK

MAPKK(pSite2~u), MAPKKK(pSite~p,kk) \
    -> MAPKK(pSite2~u!1), MAPKKK(pSite~p, kk!1) \
```

```
    @ 'BIND' # binding MAPKKK-P and MAPKK
MAPKK(pSite2~u!1), MAPKKK(kk!1) -> MAPKK(pSite2~u), MAPKKK(kk) \
    @ 'BREAK' # unbinding MAPKKK-P and MAPKK
MAPKK(pSite2~u!1), MAPKKK(kk!1) -> MAPKK(pSite2~p), MAPKKK(kk) \
    @ 'MODIFY' # phosphorylation of MAPKK

MAPKK(pSite1~p), MAPKK-Pase(kk) -> MAPKK(pSite1~p!1), MAPKK-Pase(kk!1) \
    @ 'BIND' # binding MAPKK-Pase and MAPKK-P/PP
MAPKK(pSite1~p!1), MAPKK-Pase(kk!1) -> MAPKK(pSite1~p), MAPKK-Pase(kk) \
    @ 'BREAK' # unbinding MAPKK-Pase and MAPKK-P/PP
MAPKK(pSite1~p!1), MAPKK-Pase(kk!1) -> MAPKK(pSite1~u), MAPKK-Pase(kk) \
    @ 'MODIFY' # dephosphorylation of MAPKK-P/PP

MAPKK(pSite2~p), MAPKK-Pase(kk) -> MAPKK(pSite2~p!1), MAPKK-Pase(kk!1) \
    @ 'BIND' # binding MAPKK-Pase and MAPKK-P/PP
MAPKK(pSite2~p!1), MAPKK-Pase(kk!1) -> MAPKK(pSite2~p), MAPKK-Pase(kk) \
    @ 'BREAK' # unbinding MAPKK-Pase and MAPKK-P/PP
MAPKK(pSite2~p!1), MAPKK-Pase(kk!1) -> MAPKK(pSite2~u), MAPKK-Pase(kk) \
    @ 'MODIFY' # dephosphorylation of MAPKK-P/PP

### MAPK ###

MAPK(pSite1~u), MAPKK(k,pSite1~p,pSite2~p) \
    -> MAPK(pSite1~u!1), MAPKK(k!1,pSite1~p,pSite2~p) \
    @ 'BIND' # binding MAPKK-PP and MAPK
MAPK(pSite1~u!1), MAPKK(k!1) -> MAPK(pSite1~u), MAPKK(k) \
    @ 'BREAK' # unbinding MAPKK-PP and MAPK
MAPK(pSite1~u!1), MAPKK(k!1) -> MAPK(pSite1~p), MAPKK(k) \
    @ 'MODIFY' # phosphorylation of MAPK

MAPK(pSite2~u), MAPKK(k,pSite1~p,pSite2~p) \
    -> MAPK(pSite2~u!1), MAPKK(k!1,pSite1~p,pSite2~p) \
    @ 'BIND' # binding MAPKK-PP and MAPK
MAPK(pSite2~u!1), MAPKK(k!1) -> MAPK(pSite2~u), MAPKK(k) \
    @ 'BREAK' # unbinding MAPKK-PP and MAPK
MAPK(pSite2~u!1), MAPKK(k!1) -> MAPK(pSite2~p), MAPKK(k) \
    @ 'MODIFY' # phosphorylation of MAPK

MAPK(pSite1~p), MAPK-Pase(k) -> MAPK(pSite1~p!1), MAPK-Pase(k!1) \
    @ 'BIND' # binding MAPK-Pase and MAPK-P/PP
MAPK(pSite1~p!1), MAPK-Pase(k!1) -> MAPK(pSite1~p), MAPK-Pase(k) \
    @ 'BREAK' # unbinding MAPK-Pase and MAPK-P/PP
MAPK(pSite1~p!1), MAPK-Pase(k!1) -> MAPK(pSite1~u), MAPK-Pase(k) \
    @ 'MODIFY' # dephosphorylation of MAPK-P/PP

MAPK(pSite2~p), MAPK-Pase(k) -> MAPK(pSite2~p!1), MAPK-Pase(k!1) \
    @ 'BIND' # binding MAPK-Pase and MAPK-P/PP
MAPK(pSite2~p!1), MAPK-Pase(k!1) -> MAPK(pSite2~p), MAPK-Pase(k) \
    @ 'BREAK' # unbinding MAPK-Pase and MAPK-P/PP
MAPK(pSite2~p!1), MAPK-Pase(k!1) -> MAPK(pSite2~u), MAPK-Pase(k) \
    @ 'MODIFY' # dephosphorylation of MAPK-P/PP
```

4 Exercises

Now that you have a working Kappa model, attempt the following exercises (from Russ Harmer's tutorial on RuseBase.org)

1. **Investigate how varying the strength of the input `E1` affects the steady state of `MAPK(pSite1~p?,pSite2~p?)`. Plot a rough dose-response curve.**

2. **Investigate the amplification that the cascade can produce: how much more `MAPKK` can you have than `MAPKKK`; or `MAPK` than `MAPKK`?**

3. **How many agents can a given `MAPKK` be bound to at any given time? Modify the rules so that it can bind at most one other agent at a time. How does this affect the dynamics of the system?**

4. **How can a given `MAPKK`, with sites `pSite1` and `pSite` both in state `u`, become 'doubly phosphorylated', i.e. sites `pSite1` and `pSite` both in state `p`? As written, the rules define a 'distributive' cascade; modify the rules to 'processive' cascade where a single `MAPKK` can become doubly phosphorylated through the action of a single `MAPKKK` (with no need for unbinding in between phosphorylations). How does this affect the time course (and dose-response) curve of the system?**

5. **Add new rules to the model implementing a negative feedback from doubly-phosphorylated `MAPK` to `E1`. Can you find rate constants for these rules that make the dynamics of the system oscillate? or produce a pulse response?**