The Microsoft Research - University of Trento
Centre for Computational
and Systems Biology

Technical Report CoSBi 13/2007

A formal and integrated framework to simulate evolution of biological pathways

Lorenzo Dematté
*The Microsoft Research - University of Trento
Centre for Computational and Systems Biology*
dematte@cosbi.eu

Corrado Priami
*The Microsoft Research - University of Trento
Centre for Computational and Systems Biology*
priami@cosbi.eu

Alessandro Romanel
*The Microsoft Research - University of Trento
Centre for Computational and Systems Biology*
romanel@cosbi.eu

Orkun S. Soyer
*The Microsoft Research - University of Trento
Centre for Computational and Systems Biology*
soyer@cosbi.eu

*This is the preliminary version of a paper that will appear in
Theoretical Computer Science
available at* http://www.elsevier.com/wps/find/journaldescription.cws_home/505625/description#description

Abstract

We present a formal approach to the study of evolution of biological pathways. The basic idea is to use the Beta Workbench to model and simulate pathway in connection with evolutionary algorithms to implement mutations. A fitness function is used to select individuals at any generation. The feasibility of the approach is demonstrated on the MAPK pathway.

1 Introduction

Biological pathways are studied using several methods, including but not limited to analysis of their single components, reconstruction of their series of chemical reactions and simulation of their dynamics.

Many approaches include modelling techniques to represent and analyse dynamics of complex pathways. In particular, *process calculi* have been recently applied to model and simulate biological systems [1]. The main activities up to now have been oriented to define new primitives and to show their feasibility for describing biological phenomena. Some simulation engines have been developed especially based on the stochastic π-calculus [2] like SPiM [3] or BIOSPI [4], and recently on an extension of the Beta-binders language [5][1].

Recently, there is an interest in using *evolutionary approaches* to study pathways. Understanding how pathways emerged during evolution can help us to understand their basic properties, such as the role of complexity and the importance of topology and feedback loops. Existing approaches to study evolution are commonly based on comparative genomics or proteomics and on phylogenetic analysis [6, 7]. These studies compare pathways from different organisms to see how evolution affects the internal structure of the network of interactions.

An alternative approach is to simulate evolution *in silico*, using an algorithm that mimic this process, as in [8, 9, 10, 11]. However, these approaches use ad-hoc tools and representations of pathway dynamics, while current available tools to model and simulate pathway dynamics, discussed above, do not allow for evolutionary simulations.

On the other hand, tools that mimic evolution are also known in the Computer Science domain. However, these tools and algorithms are designed for specific tasks, such as machine learning and optimization or search problems [12], and therefore they lost their strict connection with biology and they are no more adequate to study biological evolution in a realistic way.

Here, we aim at developing a specific framework to allow straightforward study of pathway evolution based on the Beta Workbench (BetaWB) [5].

[1]Available at url `http://www.cosbi.eu/Rpty_Soft_BetaWB.php`

The great flexibility of the Beta-binders process calculus in the definition of the structure of proteins allows us to introduce interesting primitives for mutations used to build domain-based interaction and mutation models. Moreover, pathway dynamics can be easily modelled, and the interactions of emerged pathways analysed. Our formal framework aims at extending the features of the BetaWB to obtain a full integration with the existing software environment. In this paper we describe the novelty of the approach unravelling the formal theory on which it is based, and we show the feasibility of the proposed solution to study evolution of pathways.

The paper is organised as follows. In the next section we briefly recall the language and the software tool BetaWB on top of which we built our work for simulating the evolution of biological pathways. The model we used to describe the general structure of signalling pathways is also introduced and explained. Sect. 3 discusses the evolutionary framework, the integration of the BetaWB simulation engine within an evolutionary algorithm and how mutations are performed on the model. An example of application of the whole framework is given in Sect. 4, and conclusions and future work directions close the paper.

2 Description of pathway dynamics

To study biological evolution of pathways, we need to describe pathway dynamics and an algorithm to simulate pathway evolution from a generation to the next one. We start describing the modeling and simulation tool.

2.1 The Beta Workbench

The Beta Workbench is a software framework for modelling and simulating biological processes [5]. It incorporates a language, a compiler to a stochastic abstract machine, the execution environment and some graphical interface components. The BetaWB language is based on β-binders [13], a process calculus developed for better representing the interactions between biological entities.

2.1.1 The Language

A BetaWB program, called also β-system, is a tuple $Z=\langle B, E, \xi\rangle$ made up of a *bio-process* B, a list of *events* E and *ambient* ξ. The bio-process B intuitively represents the structure of the system, that is a set of entities interacting in the same environment, E represents the list of possible events enabled on the system and the ambient ξ contains information about the environment, like the set T of the considered *types* (ranged over by Δ, Γ_0,

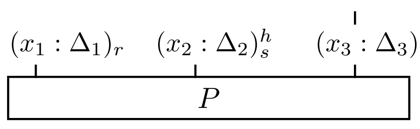
$\Sigma', \cdots$), a function $\rho : \mathcal{N} \to \mathbb{R}$ that associates stochastic rates[2] to names in $\mathcal{N}$ and the function $\alpha : T^2 \to \mathbb{R}^3$, which describes the affinity relation between couples of types. In particular, given two types Δ and Γ, the application of $\alpha(\Delta, \Gamma)$ returns a triple of stochastic rates (r, s, t), where the value r, denoted with $\alpha_c(\Delta, \Gamma)$, represents the *complexation rate*, the value s, denoted with $\alpha_d(\Delta, \Gamma)$, represents *decomplexation rate* and the value t, denoted with $\alpha_i(\Delta, \Gamma)$, represents the *inter-communication rate* of the two entities exposing the types Δ and Δ.

The syntax of the BetaWB language is the following.

$$\begin{array}{lll}
B & ::= & \mathsf{Nil} \mid \vec{B}[P] \mid B||B \\
\vec{B} & ::= & \widehat{\beta}(x, r, \Delta) \mid \widehat{\beta}(x, r, \Delta)\vec{B} \\
\widehat{\beta} & ::= & \beta \mid \beta^h \mid \beta^c \\
P & ::= & \mathsf{nil} \mid P|P \mid !\pi.P \mid M \\
M & ::= & \pi.P \mid M + M \\
\pi & ::= & x(y) \mid \overline{x}\langle y \rangle \mid (\tau, r) \mid (\mathsf{die}, r) \mid (\mathsf{ch}(x, \Delta), r) \mid \\
 & & (\mathsf{hide}(x), r) \mid (\mathsf{unhide}(x), r) \mid (\mathsf{expose}(x, s, \Delta), r) \\
\mathit{cond} & ::= & \vec{B}[P] : r \mid |\vec{B}[P]| = n \mid \vec{B}[P], \vec{B}[P] : r \\
\mathit{verb} & ::= & \mathsf{new}(\vec{B}[P], n) \mid \mathsf{split}(\vec{B}[P], \vec{B}[P]) \mid \mathsf{join}(\vec{B}[P]) \mid \mathsf{delete} \\
\mathit{event} & ::= & \bullet \mid (\mathit{cond})\ \mathit{verb} \\
E & ::= & \mathit{event} \mid \mathit{event} :: E
\end{array}$$

where $x, y \in \mathcal{N}$, $n \in \mathbb{N}$ and $r \in \mathbb{R}^+ \cup \{\infty\}$ is a stochastic rate.

Processes generated by the non terminal symbol P are referred as *pi-processes.* Bio-processes(boxes) are defined as pi-processes prefixed by specialised binders $\vec{B}$ that represent interaction capabilities. An *elementary beta binder* $\widehat{\beta}$ has the form $\beta(x, r, \Gamma)$ (active), $\beta^h(x, r, \Gamma)$ (hidden) or $\beta^c(x, r, \Gamma)$ (complexed), where the name x is the subject of the beta binder and Γ represents the type of x. Although types can be any general structure for which exist a decidable equality relation, without loss of generality we assume here that types are names taken from a countably infinit set T such that $T \cap \mathcal{N} = \emptyset$. A bio-process B is either the deadlock beta-process *Nil* or a parallel composition of boxes $\vec{B}[P]$. A box $\vec{B}[P]$ models a single biological entity, where intuitively the beta binders list $\vec{B}$ represents the interface of the entity, while the pi-process P represents its internal structure or process-unit. Moreover, the language is provided with a graphical representation of boxes:



[2]A stochastic rate is the single parameter defining an exponential distribution that drives the stochastic behaviour of an action. The rate ∞ is used to denote immediate actions, i.e., actions that are executed as soon as they become enabled.

The pairs $x_i : \Delta_i$ represent the sites through which the box may interact with other boxes. *Types* Δ_i express the interaction capabilities at x_i. The value r represents the stochastic rate associated to the name x inside the box, h represents the hidden status and the black line over the last beta binder represents the *complexed* status.

The dynamics of the system is formally specified through the *operational semantics* of the language available in [5], which uses a notion of structural congruence $\equiv$. Intuitively, that two β-systems $Z{=}\langle B, E, \xi\rangle$ and $Z'{=}\langle B', E', \xi'\rangle$ are structurally congruent ($Z \equiv Z'$), if their bio-processes B and B' and their list of events E and E' are identical up to structure ($B \equiv_b B'$ and $E \equiv_e E'$) and their ambients are equal ($\xi{=}\xi'$). Moreover, two boxes representing interacting entities, are considered of the same species only if they are structurally congruent. Given a β-system, its temporal evolution is described by three types of actions.

Monomolecular actions describe the dynamics of single boxes. More precisely, an *intra-boxes communication* allows components to interact within the same box, the *expose* action adds a new site of interaction to the interface of the box containing the expose, the *change* action modifies the type of an interaction site, *hide* and *unhide* actions make respectively invisible and visible an interaction site. Finally, the *die* action eliminates the box that performs the action and, recursively, all the boxes directly or indirectly complexed with them.

Bimolecular actions describe interactions that involves two boxes. The *complex* operation creates a dedicated communication binding between boxes over compatible and unhidden elementary beta binders, while the *decomplex* operation destroys an already existing dedicated binding. The stochastic rates associated to complex and decomplex operations are, respectively, the complexation and decomplexation rates derived from the affinity function. In the example, the complexation rate is $\alpha_c(\Delta, \Gamma)$, while the decomplexation rate is $\alpha_d(\Delta, \Gamma)$. The information about the existing dedicated bindings is maintained in the environment. The last bimolecular action is the *inter-boxes communication*, which enables interaction between boxes over compatible and unhidden elementary beta binders. Suppose Δ and Γ be the types associated to the involved elementary beta binders. If $\alpha_c(\Delta, \Gamma) > 0$, then the *inter-communication* is enabled, with rate $\alpha_i(\Delta, \Gamma)$, only after a dedicated communication binding, over the involved beta binders, has been created by a *complex* operation. Otherwise, the *inter-boxes communication* is simply enabled with rate $\alpha_i(\Delta, \Gamma)$.

An **Event** is the composition of a condition *cond* and an action *verb* and it is triggered only when the condition associated with the event is satisfied. Events can be considered as global rules of the system which can substitute, create and delete boxes from the system. In particular, the list E can contain five types of events: *join*, which substitutes two boxes with single ones; *split*, which substitutes a box with two boxes; *new*, which introduces a specified number of instances of a box; *delete*, which eliminates boxes.

Definition 1 *The BetaWB* Stochastic Transition System *(*STS*) is referred as $\mathcal{S} = (\mathcal{Z}, \xrightarrow{r}_s, Z_0)$, where $\mathcal{Z}$ is the set of β-systems, $Z_0 \in \mathcal{Z}$ is the initial β-system and $\xrightarrow{r}_s \subseteq \mathcal{Z} \times \mathbb{R} \times \mathcal{Z}$ is the stochastic transition relation, where r is a stochastic rate constant and is derived using information in the syntax and in the ambient of the β-system.*

For a more detailed and formal description of the language, its operational semantics and the laws of its structural congruence relation we refer the reader to [5].

2.1.2 Simulator and Companion Tools

The BetaWB simulator is the core part of the Beta Workbench and it is built as a composition of three logical blocks: the *compiler*, the *runtime environment* and the *stochastic simulation engine*.

The BetaWB simulator is a command-line application that takes as input two text files that describe an initial β-system $Z{=}\langle B, E, \xi\rangle$. The first file describes the bio-process B and the list of events E with a simple functional-like syntax, while the second one contains the specification of the set of types T, the affinity function α and the function ρ.

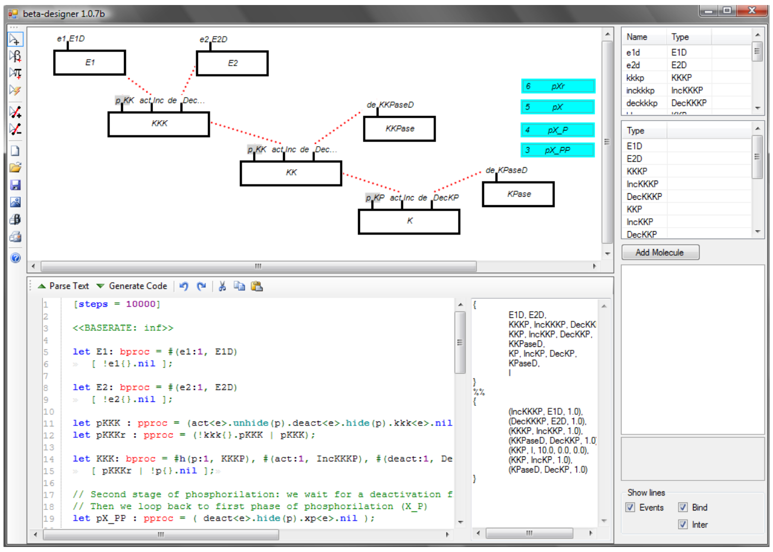
The *compiler* translates these files into a runtime representation that is then stored into the *runtime environment*. The *runtime environment* provides the *stochastic simulation engine* with primitives for checking the current state of the system and for modifying it. The *stochastic simulation engine* handles the time evolution of the environment in a stochastic way and preserves the semantics of the language. The stochastic simulation engine implements an efficient variant of the Gillespie's algorithm described in [14].

The BetaWB designer is a graphical tool that allows the user to write β-system in a graphical way (Fig.1). In particular, it is possible to draw boxes, pi-processes, interactions, events and to form complexes using graphs. The graphical format and the textual description of the β-system are interchangeable: the tool can parse and generate the graphical representation from any valid β-system, and generate the textual representation from the graphical form. The textual representation can then be used as input of the BetaWB simulator.

The BetaWB plotter is a graphical tool that parses and display simulation outputs.

2.2 A compositional model for signalling pathways

A *signalling pathway* is any biological process that converts one kind of signal or stimulus into another; this conversion is also called *signal transduction*. In general, a signalling pathway results in a composition or cascade



Name	Type
e1d	E1D
e2d	E2D
kkkp	KKKP
inckkkp	IncKKKP
deckkkp	DecKKKP

Type
E1D
E2D
KKKP
IncKKKP
DecKKKP
KKP
IncKKP
DecKKP

```
[steps = 10000]

<<BASERATE: inf>>

let E1: bproc = #(e1:1, E1D)
»   [ !e1{}.nil ];

let E2: bproc = #(e2:1, E2D)
»   [ !e2{}.nil ];

let pKKK : pproc = (act<e>.unhide(p).deact<e>.hide(p).kkk<e>.nil
let pKKKr : pproc = (!kkk{}.pKKK | pKKK);

let KKK: bproc = #h(p:1, KKKP), #(act:1, IncKKKP), #(deact:1, De
»   [ pKKKr | !p{}.nil ];»

// Second stage of phosphorilation: we wait for a deactivation f
// Then we loop back to first phase of phosphorilation (X_P)
let pX_PP : pproc = ( deact<e>.hide(p).xp<e>.nil );
```

```
{
        E1D, E2D,
        KKKP, IncKKKP, DecKK
        KKP, IncKKP, DecKKP,
        KKPaseD,
        KP, IncKP, DecKP,
        KPaseD,
        I
}
%%
{
        (IncKKKP, E1D, 1.0),
        (DecKKKP, E2D, 1.0),
        (KKKP, IncKKP, 1.0),
        (KKPaseD, DecKKP, 1.0
        (KKP, I, 10.0, 0.0, 0.0),
        (KKP, IncKP, 1.0),
        (KPaseD, DecKP, 1.0)
}
```

Figure 1: BetaWB designer: example of graphical representation.

of biochemical reactions that are carried out by proteins and linked through second messengers. Biological signal transduction allows a cell or organism to sense its environment and react accordingly. Typically, a signalling pathway has one (or more) input, represented by any environmental stimulus, and one (or more) output, represented by an active protein.

In the BetaWB language we represent a protein as a biological entity composed by a set of *sensing domains*, a set of *effecting domains* and an *internal structure*. Sensing domains are the places where the protein receives signals, effecting domains are the places that a protein uses for propagating signals, and the internal structure codifies for the mechanism that transforms an input signal into a protein conformational change, which can result in the activation or deactivation of another domain. This is inspired by the available knowledge of protein structure and function (see for example [15]).

As explained in Sec.2.1, the Beta Workbench is a general framework for modeling and simulating biological processes; each biological entity is modelled with a box, which is a composition of an interface and an internal process unit. This gives an effective way for modeling proteins by decomposing the domains of interaction and the internal structure into two different constructs. Moreover, the compositional nature of the language allows us to design and apply mutations on biological entities in an effective, simple and intuitive way.

In this application of evolutionary algorithms to β-systems we designed a general methodology for modeling proteins by providing patterns for modeling interaction domains and internal structures. Domains are represented using beta binders ($A+$ and $A-$), interaction sites with an affinity. As shown in Fig. 2 (where we omit subjects and rates of beta binders for the sake of readability and where arrows indicate that types associated to binders are compatible), the *sensing domain* is represented by a pair of binders, one for receiving a signal of *activation* (e.g. phosphorylation) and the other for receiving a signal of *deactivation* (e.g. dephosphorylation) sent to the protein. The *effecting domain* (Ao) is instead used to communicate, and so to activate or inhibit, other proteins.

Pattern of pi-processes and boxes for modeling proteins with single sensing and effecting domains are the following:

$$\begin{array}{lcl}
ps_1 & \triangleq & \overline{act}\langle\epsilon\rangle.(\mathsf{unhide}(o),\infty).\overline{deact}\langle\epsilon\rangle.(\mathsf{hide}(o),\infty).\overline{x}\langle\epsilon\rangle.\mathsf{nil} \\
pr_1 & \triangleq & !x().ps_1 \mid ps_1 \\
pss_2 & \triangleq & \overline{deact}\langle\epsilon\rangle.(\mathsf{hide}(o),\infty).\overline{xp}\langle\epsilon\rangle.\mathsf{nil} \\
ps_2 & \triangleq & !xp().(\overline{act}\langle\epsilon\rangle.(\mathsf{unhide}(o),\infty).pss_2 + \overline{deact}\langle\epsilon\rangle.\overline{x}\langle\epsilon\rangle.\mathsf{nil}) \\
pr_2 & \triangleq & !x().\overline{act}\langle\epsilon\rangle.\overline{xp}\langle\epsilon\rangle.\mathsf{nil} \mid \overline{act}\langle\epsilon\rangle.\overline{xp}\langle\epsilon\rangle.\mathsf{nil}
\end{array}$$

$$\begin{array}{lcl}
Adeact_1 & \triangleq & \beta^h(o,r,\Delta^o)\beta(act,r,\Delta^+)\beta(deact,r,\Delta^-) \\
 & & [\ pr_1 \mid !\overline{o}\langle\epsilon\rangle\ ] \\
Aact_1 & \triangleq & \beta(o,r,\Delta^o)\beta(act,r,\Delta^+)\beta(deact,r,\Delta^-) \\
 & & [\ !x().ps_1 \mid \overline{deact}\langle\epsilon\rangle.(\mathsf{hide}(o),\infty).\overline{x}\langle\epsilon\rangle.\mathsf{nil} \mid !\overline{o}\langle\epsilon\rangle\ ] \\
Adeact_2 & \triangleq & \beta^h(o,r,\Delta^o)\beta(act,r,\Delta^+)\beta(deact,r,\Delta^-) \\
 & & [\ ps_2 \mid pr_2 \mid !\overline{o}\langle\epsilon\rangle\ ] \\
Aact_2 & \triangleq & \beta(o,r,\Delta^o)\beta(act,r,\Delta^+)\beta(deact,r,\Delta^-) \\
 & & [\ !x().\overline{act}\langle\epsilon\rangle.\overline{xp}\langle\epsilon\rangle.\mathsf{nil} \mid pss_2 \mid ps_2 \mid !\overline{o}\langle\epsilon\rangle\ ]
\end{array}$$

where $\rho(x) = \rho(xp) = \infty$. The boxes $Adeact_1$ and $Aact_1$, which uses the pi-processes ps_1 and pr_1, represent respectively the deactive and active state of a protein which can be activated by a single external signal. In particular, if the box $Adeact_1$ executes an *inter-boxes communication* through the elementary beta binder $\beta(act, r, \Delta^+)$, the action $\overline{act}\langle\epsilon\rangle$ in $Adeact_1$ is consumed and immediately also the action $(\mathsf{unhide}(o), \infty)$ is consumed (because its rate is ∞). The obtained box is structurally congruent to $Aact_1$ and hence the protein has reached its active form, where the elementary beta binder $\beta(o, r, \Delta^o)$ is now unhide and the box can execute inter-communications through it. Now, if the box $Aact_1$ executes an *inter-boxes communication* through the elementary beta binder $\beta(deact, r, \Delta^-)$, the reverse mechanism is executed and the protein returns back in its deactive form $Adeact_1$.

The boxes $Adeact_2$ and $Aact_2$, which uses the pi-processes ps_2, pss_2 and pr_2, represent respectively the deactive and active state of a protein which can be activated by receiving a signal twice. The activation and deactivation mechanism is similar to the one described for single signal activation.

Obviously these patterns can be easily extended for modelling proteins with more than one sensing and effecting domains and for modeling mechanisms of activation based on the reception of more than two external signals.

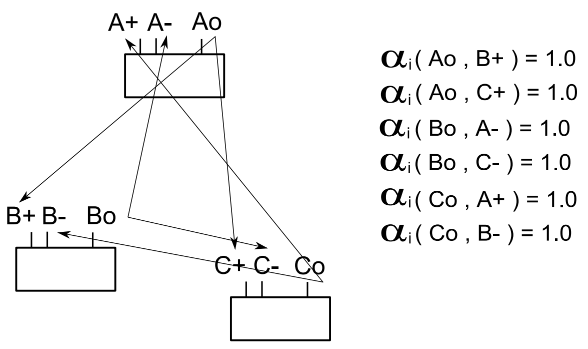

A+ A- Ao
B+ B- Bo
C+ C- Co
α_i(Ao , B+) = 1.0
α_i(Ao , C+) = 1.0
α_i(Bo , A-) = 1.0
α_i(Bo , C-) = 1.0
α_i(Co , A+) = 1.0
α_i(Co , B-) = 1.0

Figure 2: Our representation of protein interaction through sensing and effecting domains. For simplicity we omit subjects and rates of beta binders. Moreover, arrows indicate compatibility of the types, according to the α function specification reported on the right of the figure.

3 Evolutionary framework

We design a framework for simulating the evolution of pathways *in silico*. Evolution proceeds through selection acting on the variance generated by random mutation events. Individuals replicate in proportion to their performance, referred to as *fitness*. This process can be modelled as shown in Tab. 1. This algorithm differs slightly from the generic evolutionary

```
EvolutionAlgorithm ():
  $Population$ := GenerateInitialPopulation();
  for $i = 0$ to $generations$ do
    for each $Individual$ in $Population$ do
      $output$ := Simulate($Individual$);
      $fitnesses[Individual]$ := ComputeFitness($output$);
    $NewPopulation$ := ReplicateAndMutate($fitnesses, Population$);
    $Population$ := $NewPopulation$;
```

Table 1: Generic EvolutionAlgorithm.

algorithms used in computer science, being more closer to real biological observations made for the asexual reproduction of organisms.

There are four main procedures in the algorithm:

- **GenerateInitialPopulation:** the initial population can be generated randomly, from a predefined pathway configuration to be used as a starting point, or it can be a pathway with no interactions. All the individuals in the initial population can be equal at the beginning, as they will be differentiated later by the mutation phase.

- **Simulate:** each individual in the population is simulated separately using the BetaWB stochastic simulator, introduced in Sec. 2.1.

- **ComputeFitness:** the output of the simulation is used to compute the fitness value of the current individual. Note that the fitness value is problem-dependent; for an example, refer to Sec. 4.
- **ReplicateAndMutate:** this is the most important part of the algorithm; like in a real environment, individuals with the highest fitness values are more likely to survive, replicate and produce a progeny that resembles them, being not, however, completely equal to them.

```
ReplicateAndMutate ($fitnesses, Population$):
  for  $i = 0$  to  $i < Population.$Size  do
    $Individual := ChooseOneIndividual(Population, fitnesses)$;
    for each  Protein  in  Individual.Proteins  do
      if  Random() $<$ $DuplicationProbabily$  then
        $Protein2 := Protein.$Duplicate();
        $Individual.Proteins.$Add(Protein2);
      for each  Domain  in  Protein.Domains  do
        if  Random() $<$ $MutationProbability$  then
          $MutationType :=$ GetRandomMutation();
          if  IsMutationFeasible($Domain, MutationType$)  then
            $Domain2 := Individual.$PickCompatibleDomain($Domain, MutationType$);
            $Individual.$Mutate($Domain, Domain2, MutationType$);
    $NewPopulation.$Add($Individual$);
  return  $NewPopulation$;
```

Table 2: The ReplicateAndMutate algorithm.

The ReplicateAndMutate algorithm (Tab. 2) creates a new population with the same number of individuals of the current generation, using as a base the current individuals. At each step it chooses one individual, with probability proportional to its fitness (ChooseOneIndividual in the code above). This is achieved by constructing a cumulative probability array a from the *fitness* array, generating a random number in the range $0...a[Population.Size]$, and then finding the index into which the random number falls.

The selected individual will replicate and pass to the next generation. During the replication, each protein in the "genome" of the individual is given the chance to mutate, according to a probability.

A mutation is selected among all the possible types by the GetRandomMutation function, and this mutation is applied. Finally the individual, that can be equal to its predecessor or can be mutated, is added to the new population. We now define in more detail the kind of mutations that we consider in our framework.

3.1 Mutations

The different types of mutations we consider are based on real biological processes where mutations can happen at DNA and protein level. These ultimately effect pathway dynamics. For example, point mutations in a DNA sequence can change the protein amino-acid sequence, leading to changes in its tertiary structure with implications on the affinity of this protein with other proteins or substrates. Similarly, events at DNA level as gene duplication or domain shuffling can alter pathway structure and dynamics.

A computer program that wants to mimic evolution of a specie have to implement random mutations in individual during replication as well. Here, we can easily implement these molecular processes using the domain and pathway model we discussed in Sect. 2.2.

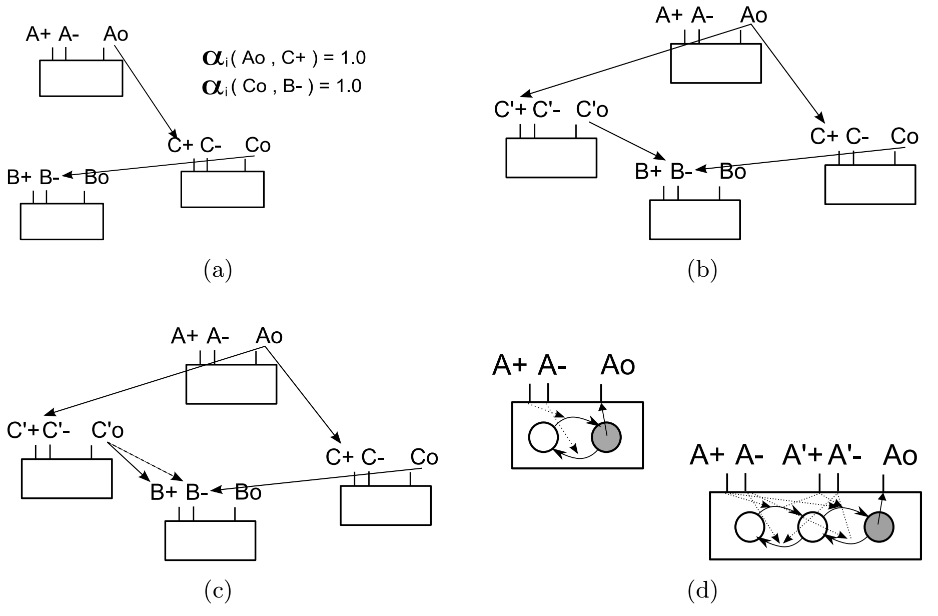


Figure 3: Different kinds of mutations: in (a) the initial configuration, displaying the α function as a list of tuples; in (b) duplication of protein C followed by mutation of domain Δ^{Co} in (c). Finally, (d) displays how the internal structure could change to accommodate the duplication of a domain.

We will take as an example the three-protein pathway represented in Fig. 3(a) and we will illustrate how different mutations are possible using the BetaWB language.

Duplication and deletion of proteins:

the implementation of this mutation consist in duplicating the associated bio-processes. Obviously the name of the bio-processes, as well as the name of the types of its binders, must be changed with fresh names. The new types will have the same interaction capabilities, i.e. the same affinities of the original types. Duplication of types is needed because subsequent mutations on one of the binders of the duplicated protein must not affect the original one. Furthermore since the new protein is a new separated entity it must not be structural equivalent to the original one. Following the model presented in Sec. 2.2, the bio-process for protein C

$$Cdeact_1 \triangleq \beta^h(o, r, \Delta^{Co})\beta(act, r, \Delta^{C+})\beta(deact, r, \Delta^{C-})[\ pr_1 \mid !\overline{o}\langle\epsilon\rangle\ ]$$

will be duplicated to

$$Cdeact_1 \triangleq \beta^h(o, r, \Delta^{Co})\beta(act, r, \Delta^{C+})\beta(deact, r, \Delta^{C-})[\ pr_1 \mid !\overline{o}\langle\epsilon\rangle\ ]$$

$$C'deact_1 \triangleq \beta^h(o, r, \Delta^{C'o})\beta(act, r, \Delta^{C'+})\beta(deact, r, \Delta^{C'-})[\ pr_1 \mid !\overline{o}\langle\epsilon\rangle\ ]$$

The same duplication will be done also for the bio-process representing the active forms of the protein. Deletion can be implemented removing the associated bio-process, or setting its cardinality to zero.

Mutation of domains:

the mutation of a domain changes the interaction capabilities of the protein to which it belongs. In our formalism, this is achieved by changing the α function on the two domains that take part in the interaction. More specifically, the mutation on a domain can be a *change of interaction*, for which we modify the affinity adding a number sampled from a normal distribution with mean zero and a little variance, an *addition of an interaction* between two domains $d1$ and $d2$, modelled as the addition of an affinity $\alpha(d1, d2) = x$, whit $x > 0$, and finally a *removal of an interaction*, between two domains $d1$ and $d2$, setting $\alpha(d1, d2) = 0$. For example, the mutation on domain oC that can be observed in Fig.3(c) is obtained by changing the α function from $\alpha(Co, B-) = 1.0$ to $\alpha(Co, B-) = 0.0$,$\alpha(Co, B+) = 0.9$

Duplication and deletion of domains:

the last possible mutation, domain duplication or deletion due to shuffling, is more complex as it requires modification of the internal behaviour in response to stimuli. Duplicating or removing domains can be easily done acting on the binders list and on the affinity function α; however, for these domains to act as sensing or effecting domains in cooperation or in antagonism with the existing ones, the internal behaviour of the process must also be changed. We devised several possible modifications of the behaviour when a domain is added, such as require communication on all the sensing domains before activating the protein (double phosphorilation, as in

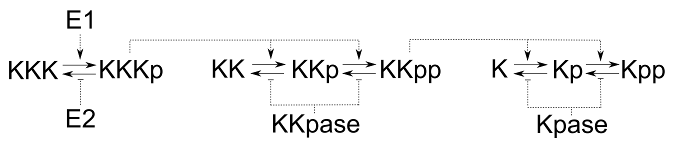


Figure 4: MAPK cascade as described in [16]. KKK denotes $MAPKKK$, KK denotes $MAPKK$ and K denotes $MAPK$. The signal $E1$ transforms KKK to $KKKp$, which in turn transforms KK to KKp to $KKpp$, which in turn transforms K to Kp to Kpp. In particular, when an input $E1$ is added, the output of Kpp increases rapidly. The transformations in the reverse direction are the result of the signal $E2$, the $KKpase$ and the $Kpase$. In particular, by removing the signal $E1$, the output level of Kpp revert back to zero.

Fig. 3(d)), different patterns of activation when more than an effecting domain exists, and so on. All these modifications can be done manipulating the abstract syntax tree of the bio-process, duplicating parts of it in a regular way. However, not all of the possible combinations of transformations have a correspondent biological meaning; a more rigorous investigation on these kind of mutations is to be done before including them in our framework and tool.

3.2 Measure of fitness

The measure of fitness is problem dependent: it varies with the kind of pathways, with the characteristics a scientist wants to investigate, and so on. This measure can be done in various ways, including stability analysis, integration of the signal, measure of the derivative. We will illustrate in our example how fitness can be computed using integration of a response.

4 An example: MAPK cascade

The mitogen-activated protein kinase cascade (MAPK cascade) is a series of three protein kinases, which is responsible for cell response to growth factors. In [16], an model for the MAPK cascade was presented (Fig.4) and analysed using ODEs; the cascade was shown to perform the function of an ultrasensitive switch and the response curves were shown to be steeply sigmoidal. A process calculi based analysis of the MAPK cascade was presented in [17]. For simplicity, in this paper we rely on a simplified version of the model, where all the enzymatic reactions of the form:

$$E + S \rightleftharpoons^{K_{ES}}_{K_{ES}^{-1}} ES \rightharpoonup^{K_{EP}} EP \rightharpoonup^{\infty} E + P$$

are substituted with simplest reactions of the form:

$$E + S \rightharpoonup^{K_{EP}} E + P$$

Using the design patterns presented in 2, a β-system for the MAPK cascade has been developed (Appendix A). Following [17], we set all the reaction rates to a nominal value of 1.0 and we initialize the system with two of $E1$, $E2$, $KKPase$ and $KPase$, 20 of KKK and 200 of KK and K. Simulating the MAPK β-system with the BetaWB simulator, similar response profiles (modulo timescale) were observed for the output of Kpp with respect to the model presented in [16], despite the differences in the simulation parameters; the system still behaves as an ultrasensitive switch (Appendix A).

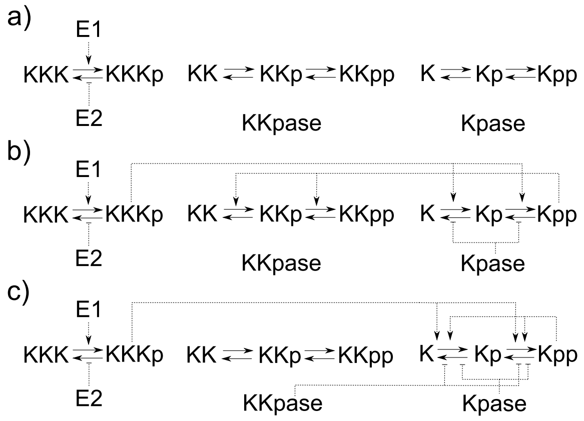
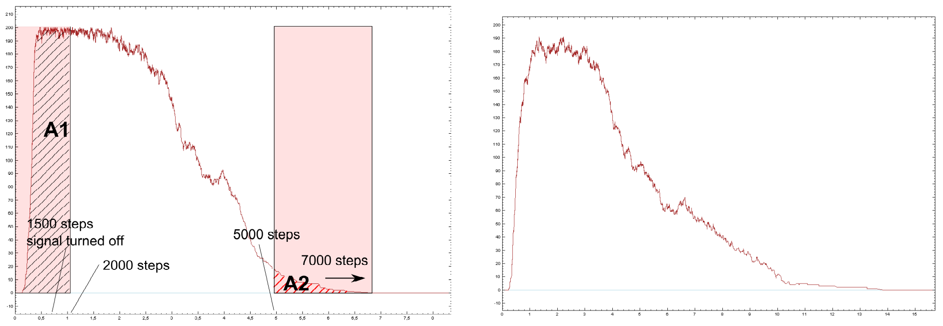


Figure 5: (a) Basic individual of the initial configuration. Only signals $E1$ and $E1$ are enabled. (b) A particular individual we obtained, with a "reverse" cascade. (c) A simil-mapk we obtained, with a good fitness value.

We use this simplified MAPK cascade β-system as a toy model for testing our evolutionary framework. In particular, we want to analyse the evolution of a population according to a fitness function which captures the essential behaviour of our MAPK cascade model.

In detail, we generate an initial population of 500 individuals containing the pathway shown in Fig.5a. This pathway resembles the observed MAPK cascade, in that it contains three kinases, two phosphatases and two signalling molecules. However, it lacks any interactions among these. In other words, we consider an ancestral organism that possessed all these proteins but lacked a signalling system similar to the MAPK cascade as observed today. The dynamic of each individual is then simulated; we run each individual for 7000 simulation steps and we remove the signal $E1$ at the step 1500. Using the output of the simulation, we then measure for each individual the corresponding fitness. The fitness function we implemented measures how rapidly the output of Kpp increases, how much the output



(a) (b)

Figure 6: (a) Time course of the Kpp concentration over the simulation time, superimposed to the integral areas for the fitness function we implemented. The fitness parameters are $i_1 = 0$, $e_1 = 2000$, $i_2 = 5000$ and $e_2 = 7000$. (b) An individual with high fitness.

of Kpp persists after removing the signal $E1$ and how rapidly the output of Kpp returns back to zero. Let $out = \{n_0, n_1, ..., n_{7000}\}$ be the tuple representing the Kpp dynamics in time of an individual, then the fitness for out is formally computed by the following formula:

$$fitness(out) = \mu + \left(\frac{\sum_{j=i_1}^{e_1} n_j}{Kpp_M * (e1 - i1)} - \left(\gamma * \frac{\sum_{j=i_2}^{e_2} n_j}{Kpp_M * (e2 - i2)}\right)\right)$$

The two sums, that we denote respectively with $A1$ and $A2$, represent discrete integrals and are normalized with respect to their possible maximum values (see Fig.6). The values i_1, e_1, i_2 and e_2 are changeable parameters that define the boundaries for the computation of the two discrete integrals present in the formula, and the value Kpp_M represents the maximum value for the Kpp respose. Moreover, μ represents the minimum fitness and γ controls the relative importance to responding to a signal and turning the response off after its removal. The reported results are for $i_1 = 0$, $e_1 = 2000$, $i_2 = 5000$, $e_2 = 7000$, $Kpp_M = 200$, $\mu = 0.1$ and $\gamma = 0.65$.

According to the algorithms presented in the previous section, the population is evolved. In this case study only *mutations of domains* are applied. Moreover, in order to maintain a biological validity for the new individuals, possible mutations are the one that satisfies the following constraints: (1) Signals $E1$ and $E2$ act only on KKK; (2) signal $E1$ and $E2$ cannot be removed; (3) a kinase can only activate other kinases or itself; (4) phosphatases are not specific but can only deactivate kinases. We iterate the evolution algorithm for 1000 generations, for different values of fitness function parameters. The dynamic for one of the obtained pathways is shown in Fig.6(b). Examples of obtained individuals are in Fig. 5, while the variation of fitness during a simulation is depicted in Fig. 7. In particular, we do not

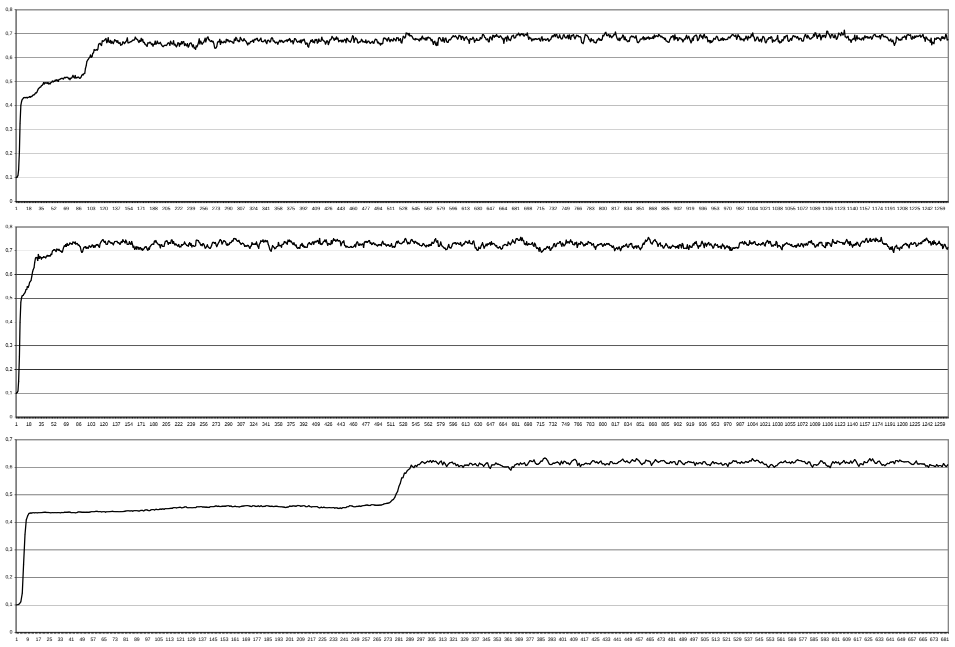


Figure 7: The modification of fitness during three different runs of the *in silico* evolution.

obtain individuals with a perfect MAPK cascade pathway, but individuals in (b) and (c) are only a reordering and a protein duplication event away from it.

Moreover, we observe three different plateaus on the average fitness variation shown by the first chart in Fig. 7. The first one is obtained after pathways evolve a kinase interaction, the second one is obtained after pathways evolve addition of a phosphatase interaction, and the third one, which persists till the final generation, is obtained after addition of other two interactions -one from a kinase and one from a phosphatase-.

5 Conclusions and Future work

In this paper we present a formal approach for simulating the evolution of pathways *in silico*. Pathway dynamics are described with the BetaWB language, a process algebra that, thanks to its biological-inspired nature, allows for modelling proteins, domains and interactions in a modular way.

We developed a modular description of signalling pathways, an evolutionary algorithm and a prototype to test our concepts. The prototype showed the potential of our approach, and the beta-binders language proved to be well-suited for this task. As presented in the small example in Sec. 4, by simulating evolution we can gain interesting insights in pathway structure and on the role of different processes.

We implemented the simpler kinds of mutations, but other interesting variations like modification of internal process structure are feasible and it will surely be a subject worth of future investigation and development.

Also, an easy to use and integrated tool for the simulation and analysis of pathway evolution under development. Finally, we plan to use the same framework also from an optimization perspective to help discovering new pathways.

References

[1] Regev, A., Shapiro, E.: Cells as computation. Nature (2002)

[2] Priami, C., Regev, A., Shapiro, E., Silvermann, W.: Application of a stochastic name-passing calculus to representation and simulation of molecular processes. Inf. Process. Lett. **80**(1) (2001) 25–31

[3] Phillips, A., Cardelli, L.: A correct abstract machine for the stochastic pi-calculus. In: Bioconcur'04, ENTCS (August 2004)

[4] Regev, A., Silverman, W., Shapiro, E.: Representation and simulation of biochemical processes using the pi-calculus process algebra. Pac Symp Biocomput (2001) 459–470

[5] Romanel, A., Dematté, L., Priami, C.: The beta workbench. Technical Report TR-03-2007, The Microsoft Research - University of Trento Centre for Computational and Systems Biology (2007)

[6] Sharan, R., Suthram, S., Kelley, R.M., Kuhn, T., McCuine, S., Uetz, P., Sittler, T., Karp, R.M., Ideker, T.: Conserved patterns of protein interaction in multiple species. Proc Natl Acad Sci **6**(102) (2005) 1974–79

[7] Babu, M., Teichmann, S., Aravind, L.: Evolutionary dynamics of prokaryotic transcriptional regulators. J. Mol. Biol. (358) (2006) 614–633

[8] Soyer, O., Bonhoeffer, S.: Evolution of complexity in signaling pathways. PNAS (103) (2006) 16337–16342

[9] Pfeiffer, T., Schuster, S.: Game-theoretical approaches to studying the evolution of biochemical systems. Trends Biochem Sci. **1**(30) (Jan 2005) 20–5

[10] Pfeiffer, T., Soyer, O., Bonhoeffer, S.: The evolution of connectivity in metabolic networks. PLoS Biol. **7**(3) (2005)

[11] P, P.F., Hakim, V.: Design of genetic networks with specified functions by evolution in silico. Proc. Natl. Acad. Sci. **2**(101) (2004) 580–5

[12] Fraser, A.S.: Simulation of genetic systems by automatic digital computers. Australian Journal of Biological Science **10** (1957) 484–499

[13] Priami, C., Quaglia, P.: Beta binders for biological interactions. In: CMSB. (2004) 20–33

[14] Gillespie, D.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. **81**(25) (1977) 2340–2361

[15] Stock, A.M., Robinson, V.L., Goudreau, P.N.: Two-component signal transduction. Annu. Rev. Biochem (69) (2000) 183–215

[16] Huang, C.Y., Ferrell, J.E.: Ultrasensitivity in the mitogen-activated protein kinase cascade. Proc Natl Acad Sci U S A **93**(19) (September 1996) 10078–10083

[17] Phillips, A., Cardelli, L., Castagna, G. Number 4230 in LNCS. In: A Graphical Representation for Biological Processes in the Stochastic pi-Calculus. Springer (2006)

AppendixA

The formal definition of the β-system of the MAPK cascade is $Z_M = \langle B, E, \xi \rangle$ where,

$$\begin{array}{rcl} B & = & \overbrace{KKK \mid\mid \cdots \mid\mid KKK}^{20} \mid\mid \overbrace{KK \mid\mid \cdots \mid\mid KK}^{200} \mid\mid \overbrace{K \mid\mid \cdots \mid\mid K}^{200} \mid\mid \\ & & E_1 \mid\mid E_1 \mid\mid E_2 \mid\mid E_2 \mid\mid Kpase \mid\mid Kpase \mid\mid KKpase \mid\mid KKpase \\ E & = & (step = 1500)\mathtt{delete}(2, E_1) :: \bullet \\ \xi & = & (T, \alpha, \rho, \emptyset) \end{array}$$

The list E contains the enabled events[3] and the boxes composing the bio-process B are defined in the following way where we recall that we are using names of pi-processes only to simplify the writing via macro-expansion:

$$\begin{array}{lcl} ps_1 & \triangleq & \overline{act}\langle\epsilon\rangle.(\mathsf{unhide}(o), \infty).\overline{deact}\langle\epsilon\rangle.(\mathsf{hide}(o), \infty).\overline{x}\langle\epsilon\rangle.\mathsf{nil} \\ pr_1 & \triangleq & !x().pS_1 \mid pS_1 \\ pss_2 & \triangleq & \overline{deact}\langle\epsilon\rangle.(\mathsf{hide}(o), \infty).\overline{xp}\langle\epsilon\rangle.\mathsf{nil} \\ ps_2 & \triangleq & !xp().(\overline{act}\langle\epsilon\rangle.(\mathsf{unhide}(o), \infty).pss_2 + \overline{deact}\langle\epsilon\rangle.\overline{x}\langle\epsilon\rangle.\mathsf{nil}) \\ pr_2 & \triangleq & !x().\overline{act}\langle\epsilon\rangle.\overline{xp}\langle\epsilon\rangle.\mathsf{nil} \mid \overline{act}\langle\epsilon\rangle.\overline{xp}\langle\epsilon\rangle.\mathsf{nil} \\ pout & \triangleq & =!\overline{out}\langle\epsilon\rangle.nil \end{array}$$

$$\begin{array}{lcl} KKK & \triangleq & \beta^h(o, r, \Delta_{KKKkase})\beta(act, r, \Delta_{incKKK})\beta(deact, r, \Delta_{decKKK}) \\ & & [\ pr_1 \mid !\overline{o}\langle\epsilon\rangle\ ] \\ KK & \triangleq & \beta^h(o, r, \Delta_{KKkase})\beta(act, r, \Delta_{incKK})\beta(deact, r, \Delta_{decKK}) \\ & & [\ ps_2 \mid pr_2 \mid !\overline{o}\langle\epsilon\rangle\ ] \\ K & \triangleq & \beta^h(o, r, \Delta_{Kkase})\beta(act, r, \Delta_{incK})\beta(deact, r, \Delta_{decK}) \\ & & [\ ps_2 \mid pr_2 \mid !\overline{o}\langle\epsilon\rangle\ ] \\ Kpp & \triangleq & \beta(o, r, \Delta_{Kkase})\beta(act, r, \Delta_{incK})\beta(deact, r, \Delta_{decK}) \\ & & [\ !x().\overline{act}\langle\epsilon\rangle.\overline{xp}\langle\epsilon\rangle.\mathsf{nil} \mid pss_2 \mid ps_2 \mid !\overline{o}\langle\epsilon\rangle\ ] \\ E_1 & \triangleq & \beta^h(out, r, \Delta_{signalE1}) \\ & & [\ pout\ ] \\ E_2 & \triangleq & \beta^h(out, r, \Delta_{signalE2}) \\ & & [\ pout\ ] \\ Kpase & \triangleq & \beta^h(out, r, \Delta_{Kpase}) \\ & & [\ pout\ ] \\ KKpase & \triangleq & \beta^h(out, r, \Delta_{KKpase}) \\ & & [\ pout\ ] \end{array}$$

The set of types is

$T = \{\Delta_{signalE1}, \Delta_{signalE2}, \Delta_{KKKkase}, \Delta_{irecKKK},$

[3]Timed events, like $(step = 1500)\mathtt{delete}(2, E_1)$, have been implemented although not yet present in the actual definition of the language.

$\Delta_{drecKKK}, \Delta_{KKkase}, \Delta_{incKK}, \Delta_{decKK},$
$\Delta_{KKpase}, \Delta_{Kkase}, \Delta_{incK}, \Delta_{decK},$
$\Delta_{Kpase}\}$

and the α function is defined in the following way:

$$\begin{array}{ll} \alpha_i(\Delta_{irecKKK}, \Delta_{signalE1}) = 1.0 & \alpha_i(\Delta_{KKpase}, \Delta_{decKK}) = 1.0 \\ \alpha_i(\Delta_{drecKKK}, \Delta_{signalE2}) = 1.0 & \alpha_i(\Delta_{KKkase}, \Delta_{incK}) = 1.0 \\ \alpha_i(\Delta_{KKKkase}, \Delta_{incKK}) = 1.0 & \alpha_i(\Delta_{Kpase}, \Delta_{decK}) = 1.0 \end{array}$$

and the function ρ is such that $\rho(x) = \rho(xp) = \infty$.

The corrisponding source code for the `BetaWB` simulator is contained in two files. The program file, that contains the structure of the system:

```
[steps = 7000]

<<BASERATE: inf>>

/* Definition of the signal E1 */
let E1: bproc = #(e1:1, signalE1)
 [ !e1{}.nil ];

/* Definition of the signal E2 */
let E2: bproc = #(e2:1, signalE2)
 [ !e2{}.nil ];

/* Phospatase for K */
let KPase: bproc = #(de:1, Kpase)
 [ !de{}.nil ];

/* Phospatase for KK */
let KKPase: bproc = #(de:1, KKpase)
 [ !de{}.nil ];

/* Definition of the pi-processes for the KKK entity */
let pKKK : pproc = (act<e>.unhide(p).deact<e>.hide(p).kkk<e>.nil);
let pKKKr : pproc = (!kkk{}.pKKK | pKKK);

/* Definition of the pi-processes for the KK and K entities */
let pX_PP : pproc = ( deact<e>.hide(p).xp<e>.nil );
let pX_P : pproc = !xp{}.(act<e>.unhide(p).pX_PP + deact<e>.x<e>.nil);
let pX : pproc = act<e>.xp<e>.nil;
let pXr : pproc = (!x{}.pX | pX);

/* definition of the deactive form of KKK */
let KKK: bproc = #h(p:1, KKKkase), #(act:1, irecKKK), #(deact:1, drecKKK)
 [ pKKKr | !p{}.nil ];

/* definition of the deactive form of KK */
let KK : bproc = #h(p:1, KKkase), #(act:1, incKK), #(deact:1, decKK)
 [ pXr | pX_P | !p{}.nil ];

/* definition of the deactive form of K */
let K : bproc = #h(p:1, Kkase), #(act:1, incK), #(deact:1, decK)
 [ pXr | pX_P | !p{}.nil ];

/* definition of the active form of K */
let KAct : bproc = #(p:1, Kkase), #(act:1, incK), #(deact:1, decK)
```

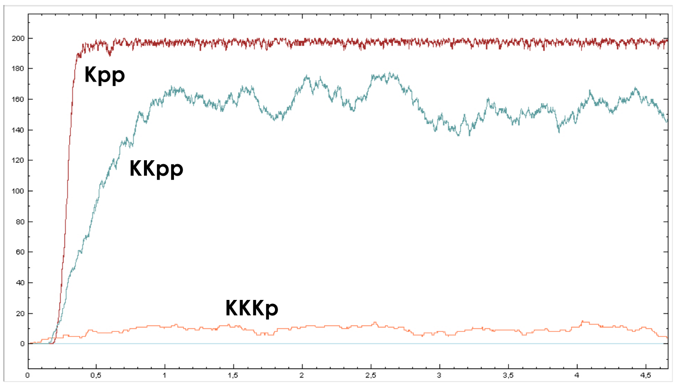


Figure 8: Result of a simulation of the MAPK cascade with the `BetaWB` simulator.

```
 [ !x{}.pX | pX_PP | pX_P | !p{}.nil ];

run 2 E1 || 2 E2 || 20 KKK || 200 KK || 2 KKPase || 200 K || 2 KPase
```

and the type file, that contains the definition of the environment:

```
{
  signalE1, signalE2,
  KKKkase, irecKKK, drecKKK,
  KKkase, incKK, decKK,
  KKpase,
  Kkase, incK, decK,
  Kpase
}
%%
{
  (irecKKK, signalE1, 1.0),
  (drecKKK, signalE2, 1.0),
  (KKKkase, incKK, 1.0),
  (KKpase, decKK, 1.0),
  (KKkase, incK, 1.0),
  (Kpase, decK, 1.0)
}
```

An example of simulation output, that shows how the system still behaves as an ultrasensitive switch, is in Fig.8.