Computational Systems Biology

Paola Quaglia

University of Trento and CoSBi

UNIVERSITÀ DEGLI STUDI DI TRENTO

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

Agenda

- Deterministic chemical kinetics
- Stochastic chemical kinetics
- Simulation: Gillespie’s direct method
- BlenX: a language for modelling system dynamics with a stochastic run-time support for simulation

Chemical kinetics: reactions

$$n_1\, Y_1 + \ldots + n_j\, Y_j \longrightarrow m_1\, X_1 + \ldots + m_k\, X_k$$

Reactions: terminology

Reactants

$$n_1 \, Y_1 + \ldots + n_j \, Y_j \longrightarrow m_1 \, X_1 + \ldots + m_k \, X_k$$

Reactions: terminology

Reactants

Products

$$n_1\ Y_1 + \ldots + n_j\ Y_j \longrightarrow m_1\ X_1 + \ldots + m_k\ X_k$$

Reactions: terminology

Reactants

Products

$$n_1 Y_1 + \ldots + n_j Y_j \longrightarrow m_1 X_1 + \ldots + m_k X_k$$

Stoichiometries

Deterministic approach

Assume we have:

- a well-stirred and fixed volume V in thermal equilibrium;
- N chemical species, each with an initial number of molecules;
- M reactions through which the species can interact.

General question:

Which will be the population levels of species after a period of time?

The deterministic approach assumes that the number of molecules of the i-th species at time t can be represented by a continuous function $X_i(t)$:

$$\mathbf{dX_i/dt = f(X_1(t), \ldots, X_N(t))}$$

Example

Lotka-Volterra prey-predator eco-system

$$Y \longrightarrow 2Y$$

$$Y + R \longrightarrow 2R$$

$$R \longrightarrow \varnothing$$

Y represents the pre**Y**, and R the predato**R**

1. prey reproduction
2. predator reproduction, favoured by feeding on preys
3. predator natural death

Deterministic formulation

Deterministic formulation:
Time evolution is a wholly predictable process, governed by a set of coupled ODEs.

In many cases time evolution can be treated as a **deterministic** and **continuous** process with an acceptable degree of accuracy, however:

- Deterministic modelling of a biological system requires the precise knowledge of molecular dynamics (precise position and velocity of each molecule). At higher level (when less details are known), the evolution is intrinsically stochastic.
- Time evolution is not really a continuous process: population levels can change only in a discrete way.

Stochastic formulation

Stochastic formulation:
Time evolution is a **random-walk** process, governed by a single stochastic differential equation (*master equation*).

The stochastic formulation has a firmer physical kinetic basis than the deterministic formulation, and is especially relevant when dealing with low concentrations.

The stochastic master equation, though, is very often mathematically intractable.

Gillespie’s Direct Method

It is a computational method (an algorithm) which takes explicit account of the fact that time evolution of spatially homogeneous systems is a **discrete** (vs. continuous) **stochastic** (vs. deterministic) process and offers an applicable alternative to the solution of the master equation.

References:

- D. T. Gillespie, J. Comput. Phys., vol. 22, 1976.
- D. T. Gillespie, J. Physical Chemistry, vol. 81, 1977.

Gillespie’s Direct Method (ctd.)

The method is implemented to answer the

General question:
If N species can interact through one of M reactions in a fixed volume, which will be the population levels of species after a period of time?

The algorithm generates a trajectory of the evolution of the systems: it calculates **which** reaction will occur next and **when** it will occur.

Gillespie’s Direct Method (ctd.)

Underlying physics:

- reactions are collisions
- molecules are randomly and uniformly distributed in the volume (assuming the system be in thermal equilibrium)

From this Gillespie argues that, although one cannot rigorously compute the number of collisions occurring in V between molecules of two given species, it is possible to precisely compute the probability of such collision occurring in any infinitesimal time interval.

Then the key point of the method is:
using ***reaction probability per unit time*** instead of ***rate constants.***

Gillespie’s Direct Method (ctd.)

Given M reactions R_1, ..., R_M, there exist M constants, which only depend on the physical properties of the involved molecules and on the temperature of the system, such that:

c_j dt = **average** probability that a particular combination of R_j reactants will react in the next infinitesimal time interval dt.

Why “average”?

h_j = number of distinct R_j molecular reactant combinations in V at time t.

$c_j h_j$ dt is the probability that an R_j reaction will occur in the next infinitesimal time interval (t, t + dt).

Gillespie’s Direct Method (ctd.)

Computing h_j is not hard:

First order reactions:

$Y \longrightarrow \ldots \quad \Rightarrow \quad h = |Y|$

Second order reactions:

$X + Y \longrightarrow \ldots \quad \Rightarrow \quad h = |X|\,|Y|$

$2\,X \longrightarrow \ldots \quad \Rightarrow \quad h = |X|\,(|X|-1)/2$

Gillespie's Direct Method (ctd.)

At time T, what we need to know to implement the next simulation step is:

- when the next reaction will occur,
- which kind of reaction it will be.

This is a probabilistic information given by:

P(t,j) dt = probability that at time t the next reaction will be a R_j reaction and will occur in the infinitesimal interval (T+t,T+t+dt)

= **P**(t,j) dt = $a_j \exp(-a_0\, t)$ $(t \geq 0)$

where $a_j = c_j\, h_j$ and $a_0 = \Sigma_{j=1..M}\, a_j$

Simulation algorithm

Initialization (set the values c_j and the population levels)

Compute $a_0 = \Sigma_{j=1..M}\, a_j$

Generate two random numbers n_1, n_2 in [0,1] and compute

- $t = (1/a_0) \ln (1/n_1)$
- j such that $\Sigma_{k=1..j-1}\, a_j < n_2\, a_0 \le \Sigma_{k=1..j}\, a_j$

Adjust population levels according to R_j and set $T=T+t$ then iterate from step 2

Applying Gillespie’s method in process calculi

Stochastic process calculi:
formal languages for interacting processes

Basic ingredients:

1. a set of elementary actions with associated rate values (meaning that the delay of the corresponding activity is a random variable with an exponential distribution)
2. a limited set of operators to specify (at least):
    - the temporal ordering of actions
    - possible coordination/interaction between actions

Applying the method in process calculi (ctd)

These formalisms are:

- scalable (to describe phenomena from biochemistry up to populations of cells);
- amenable to computer execution (analysis and/or simulation)

A very good point:

These calculi come with an operational semantics that ease the representation of process behaviours as graphs.

Example: biochemical stochastic pi-calculus (Priami, Regev, Silverman, Shapiro, 2001)

$$(\ldots + (\overline{x}\langle z\rangle, r).Q)|((x(y), r).P + \ldots) \xrightarrow{x, r_b \cdot 1 \cdot 1} Q|P\{z/y\},\ x \notin \mathcal{H}$$

$$\begin{aligned}&(\ldots + (\overline{x}\langle z\rangle, r).Q + (x(y), r).P)|\\ &\quad((\overline{x}\langle z\rangle, r).Q + (x(y), r).P + \ldots) \xrightarrow{x, 1/2 \cdot r_b \cdot 2 \cdot (2-1)} Q|P\{z/y\},\ x \in \mathcal{H}\end{aligned}$$

$$\frac{P \xrightarrow{x, r_b \cdot r_0 \cdot r_1} P'}{P|Q \xrightarrow{x, r_b \cdot r_0' \cdot r_1'} P'|Q}, \begin{cases} r_0' = r_0 + In_x(Q) \\ r_1' = r_1 + Out_x(Q) \end{cases}$$

$$\frac{P \xrightarrow{x, r_b \cdot r_0 \cdot r_1} P'}{(\nu\, x)P \xrightarrow{x, r_b \cdot r_0 \cdot r_1} (\nu\, x)P'} \qquad \frac{Q \equiv P, P \xrightarrow{x, r_b \cdot r_0 \cdot r_1} P', P' \equiv Q'}{Q \xrightarrow{x, r_b \cdot r_0 \cdot r_1} Q'}$$

more examples:
BioAmbients, Brane Calculi, Core Formal Biology, Beta-binders, Bio-PEPA, ...

Agenda

- Deterministic chemical kinetics
- Stochastic chemical kinetics
- Simulation: Gillespie’s direct method
- ➡ BlenX: a language for modelling system dynamics with a stochastic run-time support for simulation

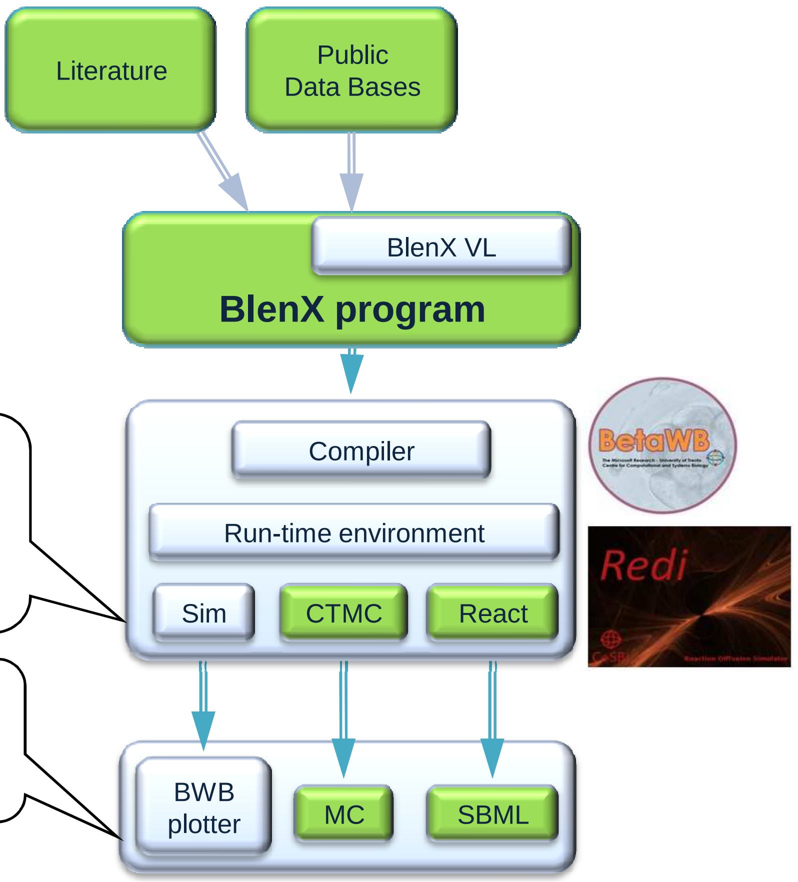
BlenX

BlenX is the kernel of a programming language based on Beta-binders (Priami and Quaglia, 2004).

In turn, BlenX is the core of CoSBi Lab.

http://www.cosbi.eu

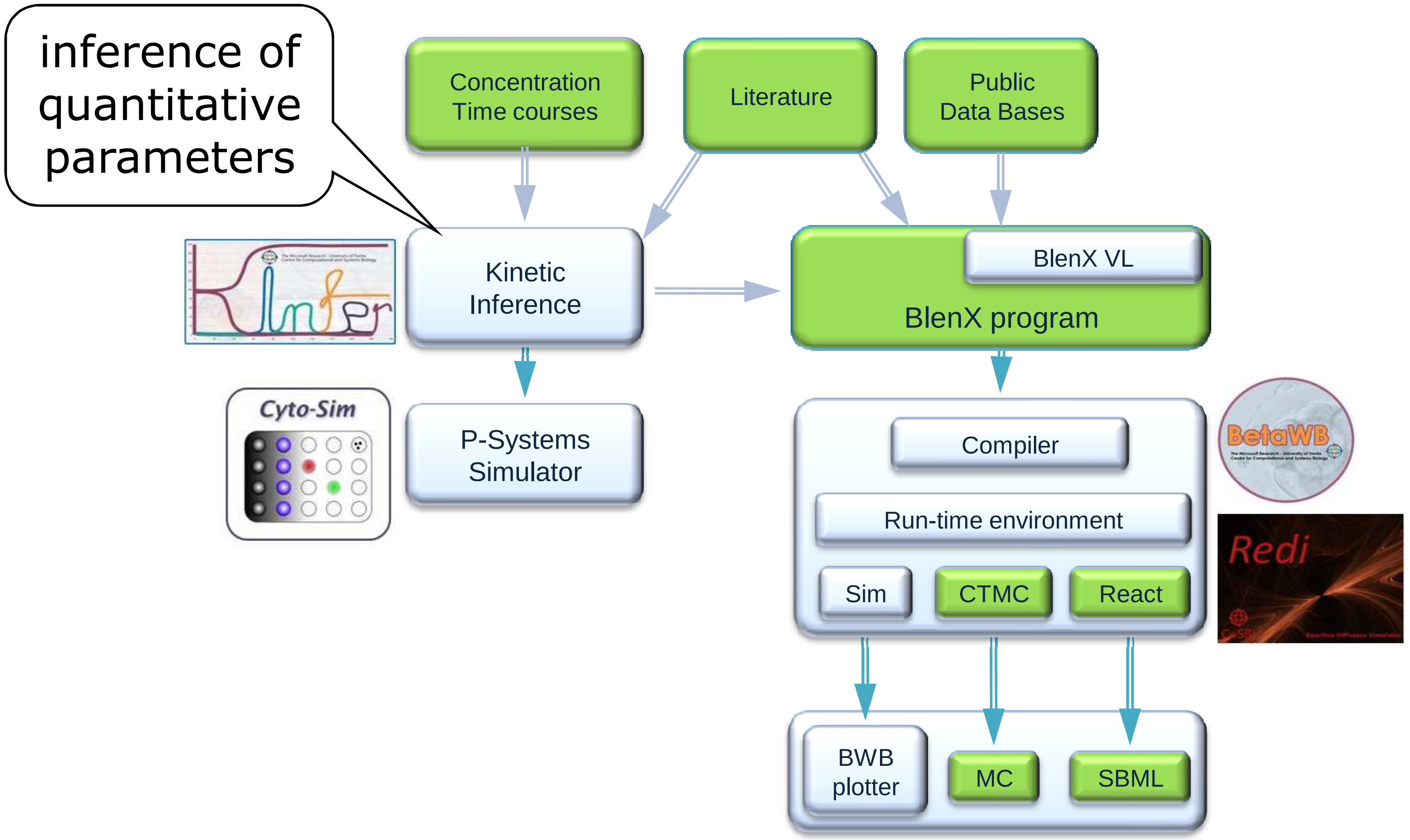
CoSBi Lab



supports both Gillespies's simulations and spatial diffusion

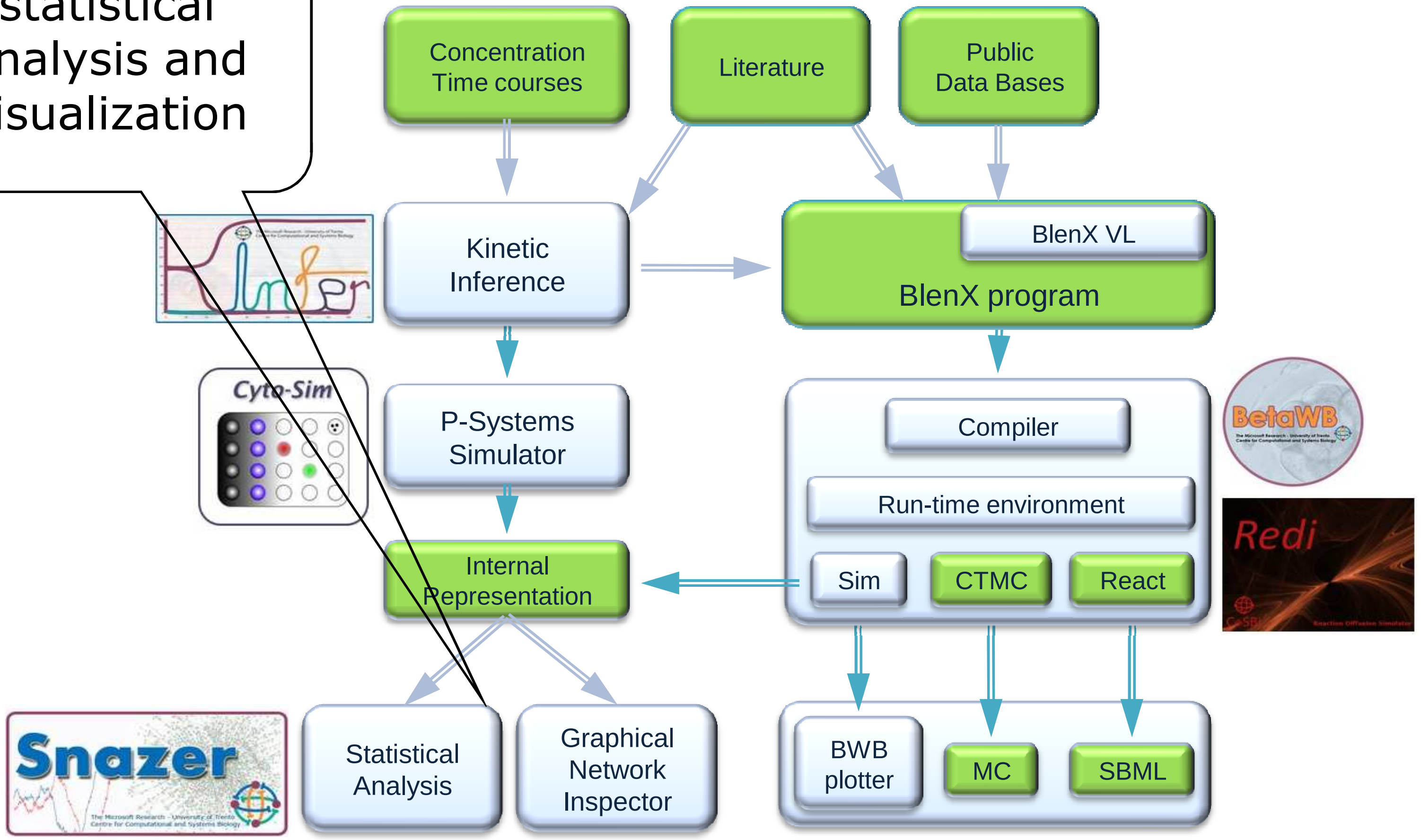
visual, Markov chain, and SBML export

CoSBi Lab


inference of quantitative parameters
Concentration Time courses
Literature
Public Data Bases
Kinetic Inference
BlenX VL
BlenX program
P-Systems Simulator
Cyto-Sim
Compiler
Run-time environment
Sim
CTMC
React
BetaWB
Redi
BWB plotter
MC
SBML

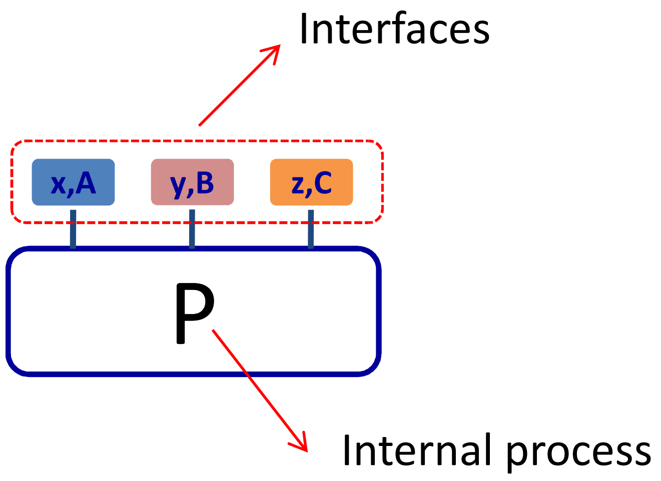
CoSBi Lab

statistical analysis and visualization



Main ingredients of BlenX

Boxes with typed interaction sites



- interaction between two boxes is allowed over “affine” interfaces, and is based on a race condition
- complexation of two boxes is driven by the affinity of the relevant sites

Biological interactions

Biological entities (mRNA, protein, …)	Boxes
Interaction capabilities (protein domains, …)	Box interaction sites & Communication
Interaction potentials	Affinity of interaction sites
Complexation	Linking boxes together into graphs
Decomplexation	Removing edges from graphs

A simple program

```
[steps = 150000]


let Y : bproc = #(y,DY)
  [ nil ];

let R : bproc = #(r,DR)
  [ nil ];

let YR : bproc = #(yr,DYR)
  [ nil ];

when (Y: : 10) split(Y,Y);
when (Y,R : : 0.01) join (YR);
when (YR : : inf) split(R,R);
when (R: : 10) delete;


run 1000 Y || 1000 R || 0 YR
```

A simple program: structure of file.prog

Preamble

```
[steps = 150000]
```

Declarations

```
let Y : bproc = #(y,DY)
  [ nil ];

let R : bproc = #(r,DR)
  [ nil ];

let YR : bproc = #(yr,DYR)
  [ nil ];

when (Y: : 10) split(Y,Y);
when (Y,R : : 0.01) join (YR);
when (YR : : inf) split(R,R);
when (R: : 10) delete;
```

Directives

```
run 1000 Y || 1000 R || 0 YR
```

A simple program: preamble

```
[steps = 150000]
```

```
let Y : bproc = #(y,DY)
  [ nil ];

let R : bproc = #(r,DR)
  [ nil ];

let YR : bproc = #(yr,DYR)
  [ nil ];

when (Y: : 10) split(Y,Y);
when (Y,R : : 0.01) join (YR);
when (YR : : inf) split(R,R);
when (R: : 10) delete;
```

```
run 1000 Y || 1000 R || 0 YR
```

Simulation information

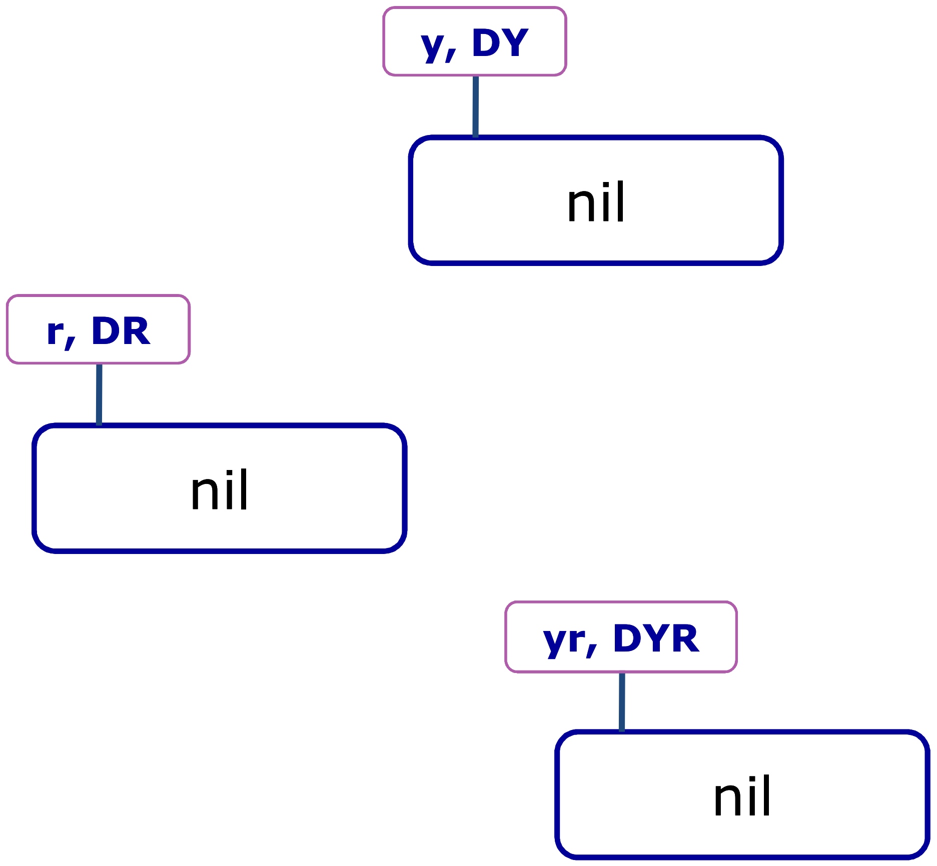
```
[ STEPS = 10000]
[ TIME = 70 ]
[ STEPS = 7, DELTA = 10 ]
```

Global stochastic rates

```
<<
    BASERATE : inf,
>>
```

A simple program: box declaration

```
[steps = 150000]

let Y : bproc = #(y,DY)
  [ nil ];

let R : bproc = #(r,DR)
  [ nil ];

let YR : bproc = #(yr,DYR)
  [ nil ];

when (Y: : 10) split(Y,Y);
when (Y,R : : 0.01) join (YR);
when (YR : : inf) split(R,R);
when (R: : 10) delete;


run 1000 Y || 1000 R || 0 YR
```



nil is the simplest internal process

A simple program: events declaration

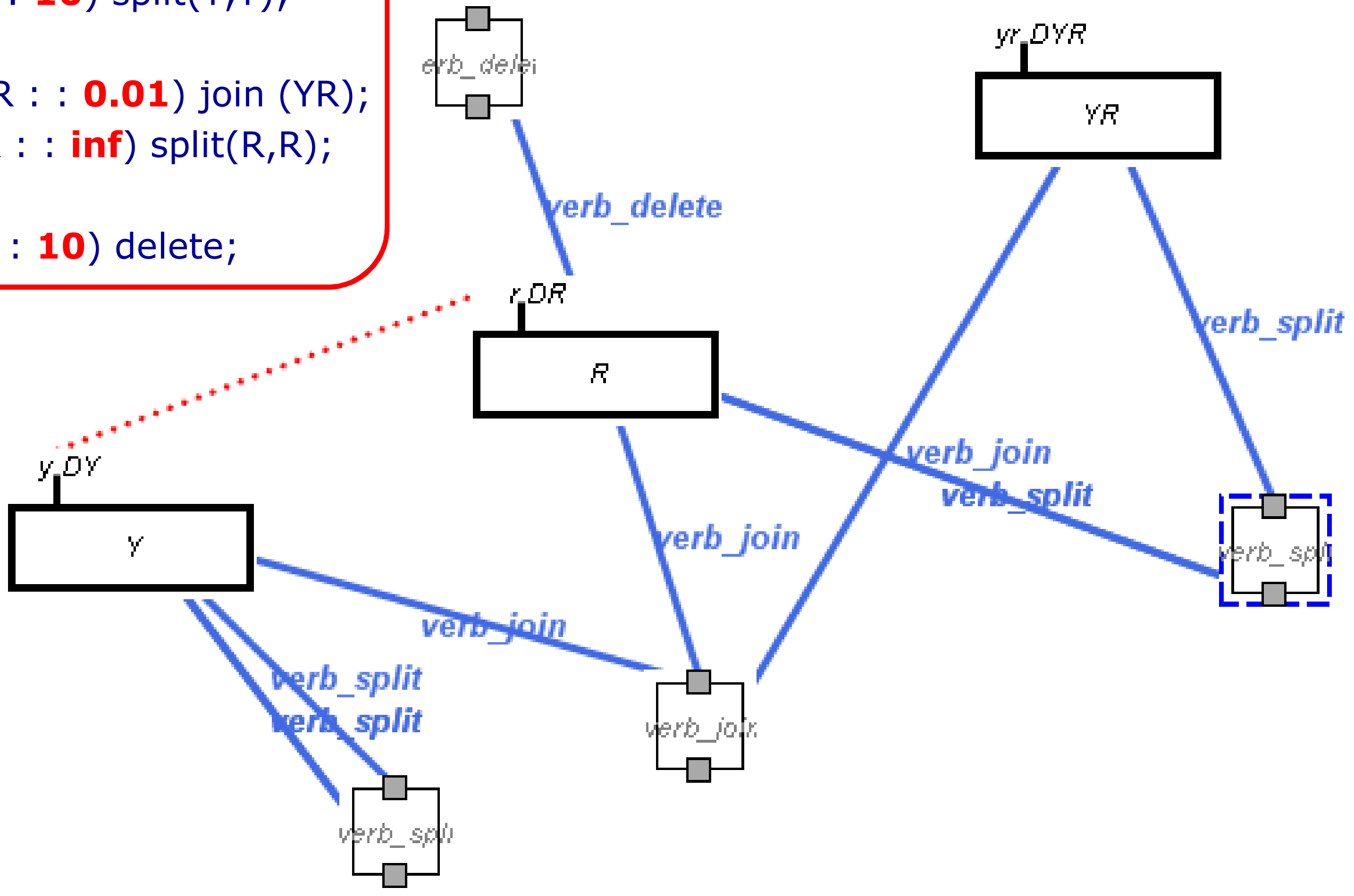
```
[steps = 150000]


let Y : bproc = #(y,DY)
  [ nil ];

let R : bproc = #(r,DR)
  [ nil ];

let YR : bproc = #(yr,DYR)
  [ nil ];

when (Y: : 10) split(Y,Y);
when (Y,R : : 0.01) join (YR);
when (YR : : inf) split(R,R);
when (R: : 10) delete;


run 1000 Y || 1000 R || 0 YR
```

Events (split, join, delete)

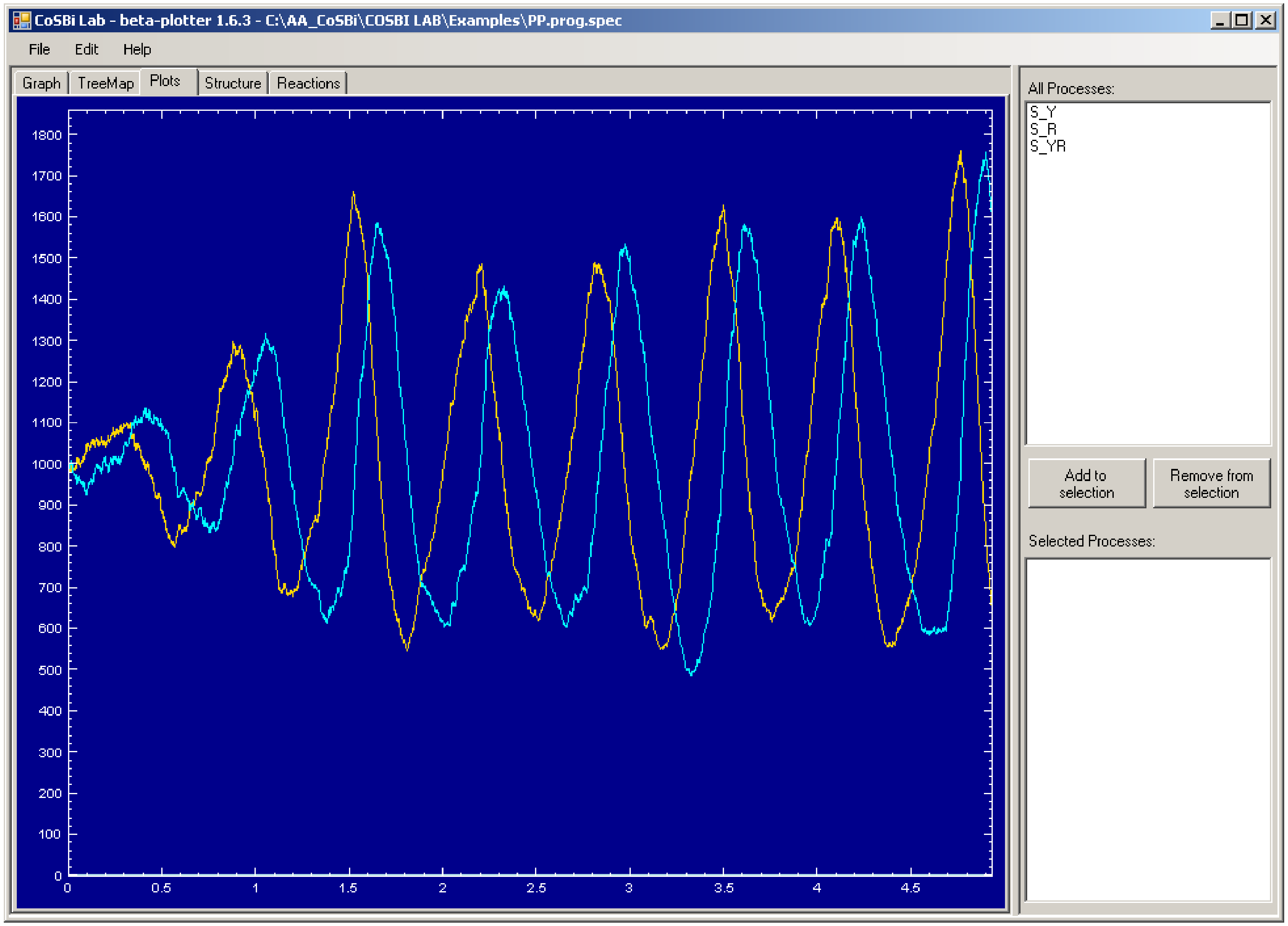
with associated rates

Lotka-Volterra, computationally

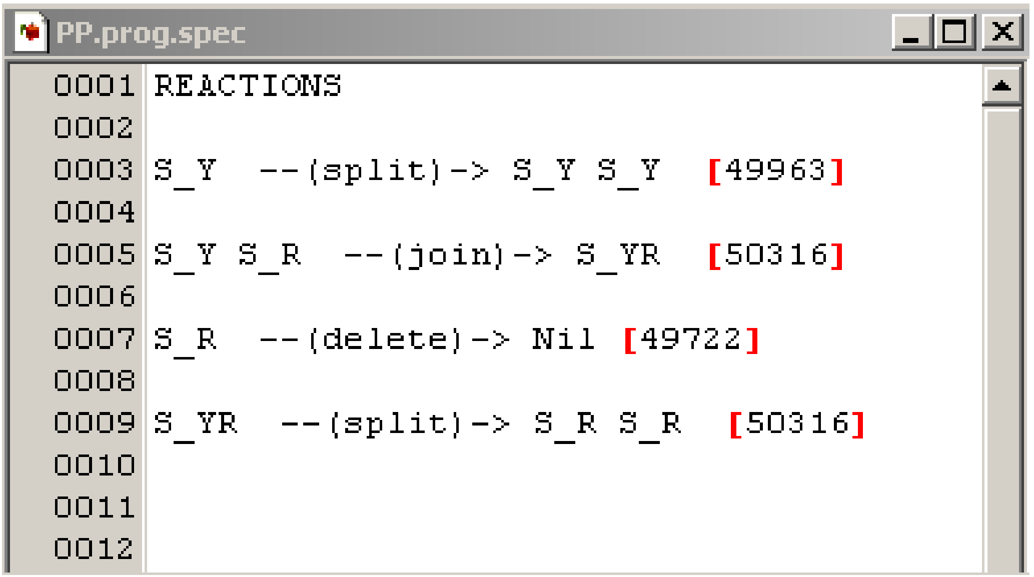
```
when (Y: : 10) split(Y,Y);

when (Y,R : : 0.01) join (YR);
when (YR : : inf) split(R,R);

when (R: : 10) delete;
```



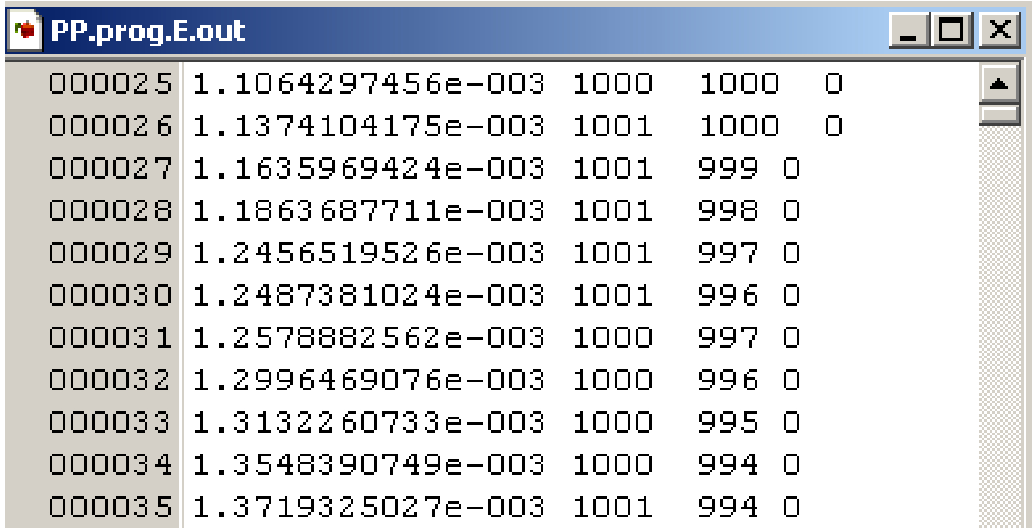
Lotka-Volterra, computationally



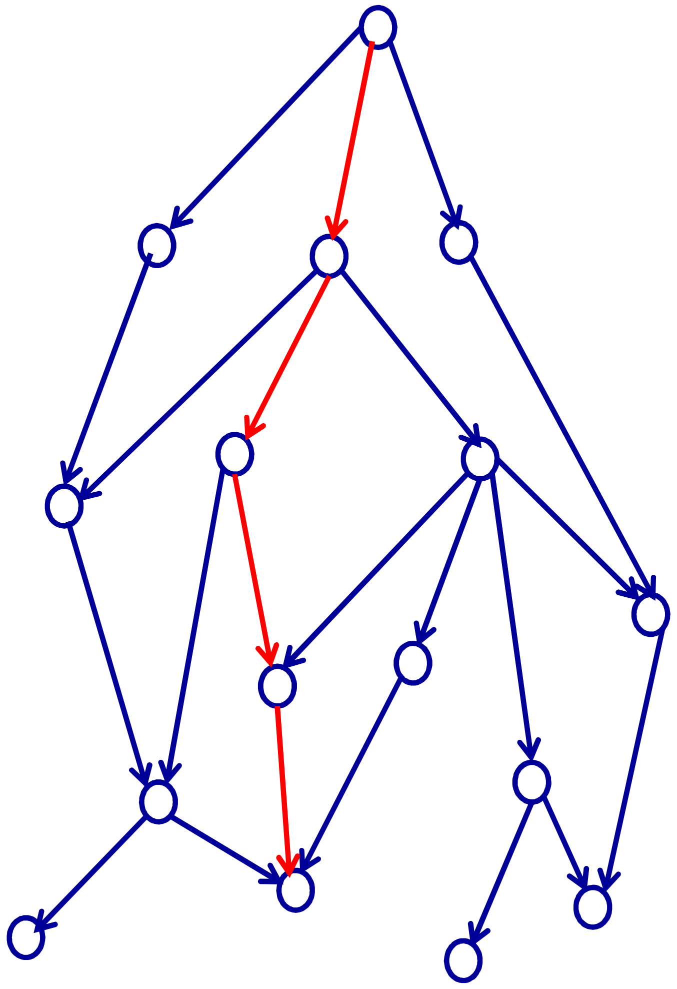
Simulation run



```
REACTIONS

S_Y  --(split)-> S_Y S_Y   [49963]

S_Y S_R  --(join)-> S_YR  [50316]

S_R  --(delete)-> Nil [49722]

S_YR  --(split)-> S_R S_R  [50316]
```



```
000025 1.1064297456e-003 1000  1000  0
000026 1.1374104175e-003 1001  1000  0
000027 1.1635969424e-003 1001  999 0
000028 1.1863687711e-003 1001  998 0
000029 1.2456519526e-003 1001  997 0
000030 1.2487381024e-003 1001  996 0
000031 1.2578882562e-003 1000  997 0
000032 1.2996469076e-003 1000  996 0
000033 1.3132260733e-003 1000  995 0
000034 1.3548390749e-003 1000  994 0
000035 1.3719325027e-003 1001  994 0
```



More than events

Communication primitives for both

- interactions between boxes, and
- interaction between parallel sub-processes within the same box

Binding and unbinding of boxes

For each model:

√ **file.prog**

file.types

Binding and unbinding: file.types

```
[steps = 150000]


let Y : bproc = #(y,DY)
  [ nil ];

let R : bproc = #(r,DR)
  [ nil ];

let YR : bproc = #(yr,DYR)
  [ nil ];

when (Y: : 10) split(Y,Y);
when (Y,R : : 0.01) join (YR);
when (YR : : inf) split(R,R);
when (R: : 10) delete;


run 1000 Y || 1000 R || 0 YR
```

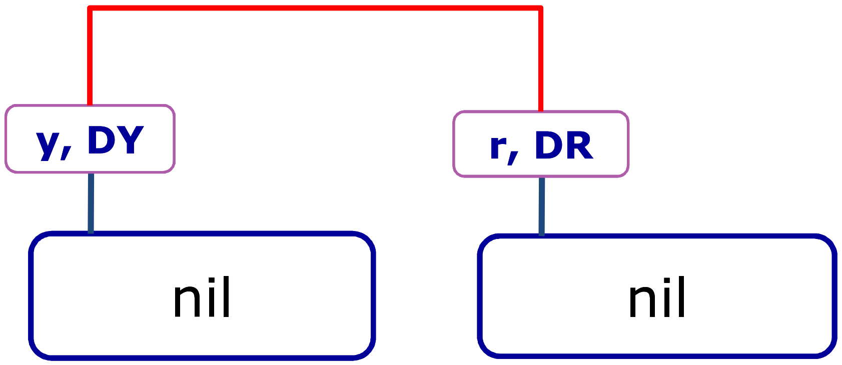
```
{
  DY,
  DR,
  DYR
}
%%
{
(DY,DR,0,0,0)
}
```

no binding (nor subsequent unbinding) is allowed between Y and R

Binding and unbinding: simple example

```
[steps = 150000]


let Y : bproc = #(y,DY)
  [ nil ];

let R : bproc = #(r,DR)
  [ nil ];

~~let YR : bproc = #(yr,DYR)~~
~~  [ nil ];~~

~~when (Y: : 10) split(Y,Y);~~
~~when (Y,R : : 0.01) join (YR);~~
~~when (YR : : inf) split(R,R);~~
~~when (R: : 10) delete;~~


run 1000 Y || 1000 R ~~|| 0 YR~~
```

```
{
  DY,
  DR~~,~~
  ~~DYR~~
}
%%
{
(DY,DR,3,2,0)
}
```



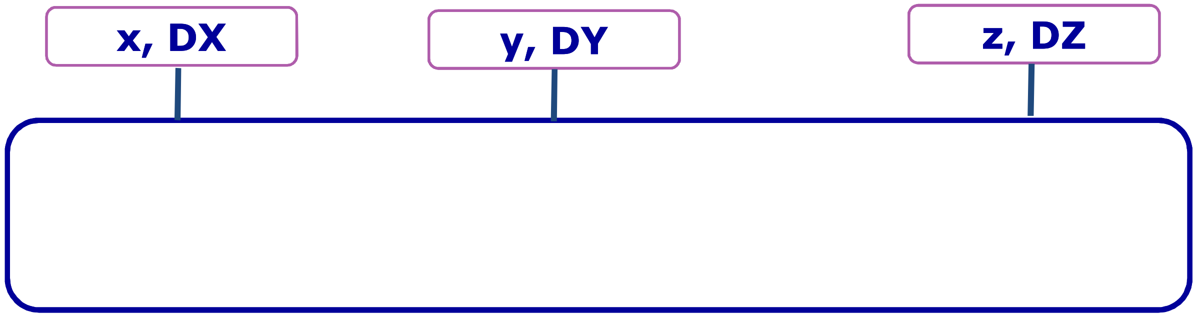
Declarations in file.prog

√ Events

√ Boxes

Internal processes

Internal processes



Interface management:

change (x, DX)

expose (w, DW)

hide (y)

Interaction management:

x!<value>. P

z?(parameter). P

u!<value>. P

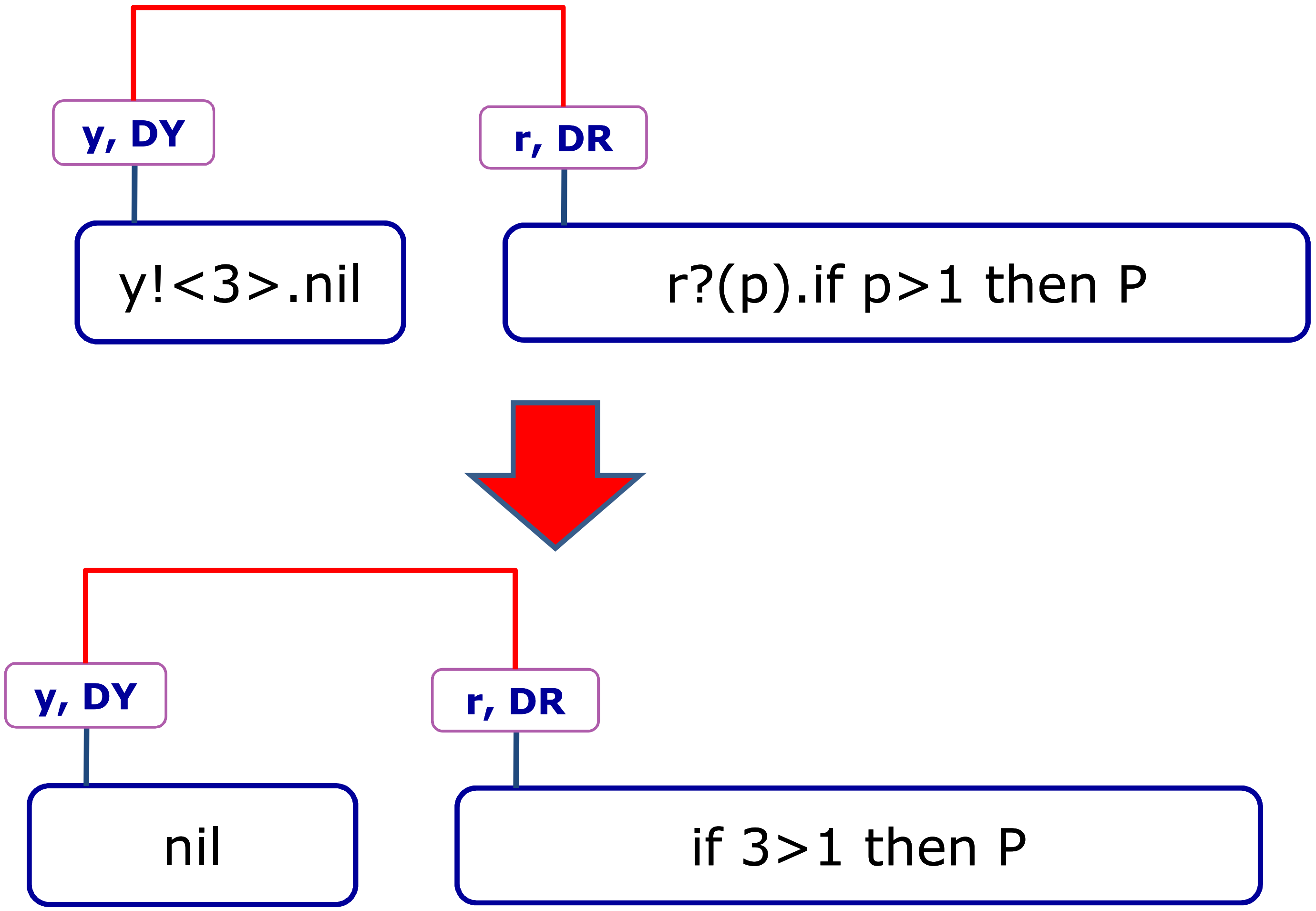
u?(parameter). P

P | Q

P + Q

if *condition* then P

Value-passing

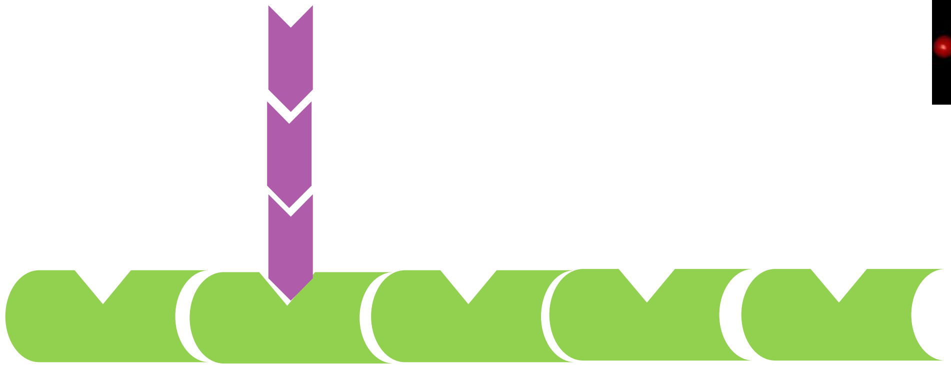


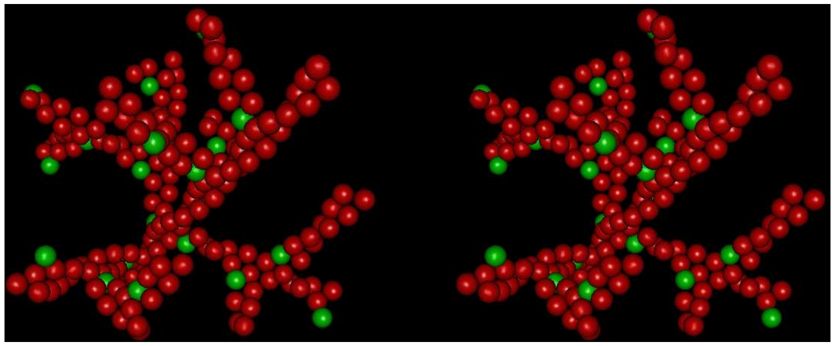
Actin polymerization

Filaments are generated from an initial feed.

Filaments can branch by complexation with ARP molecules.

A minimum distance between adjacent branches is always granted by a specific interaction **protocol**.

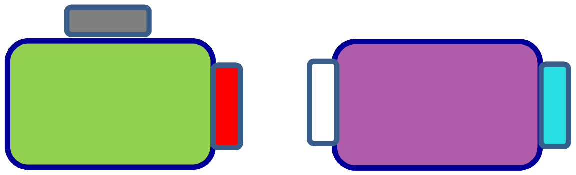




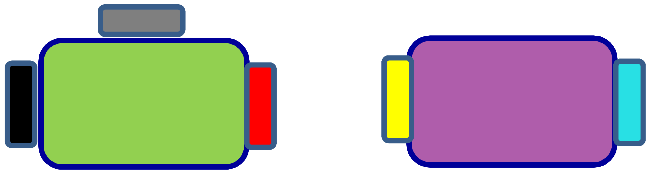
by Roberto Larcher

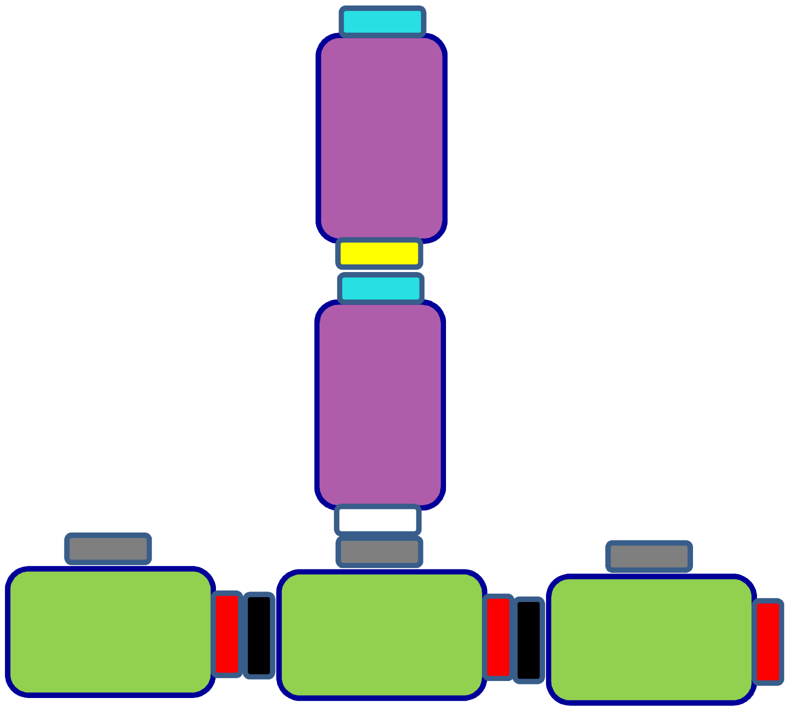
Actin polymerization

Seeds:



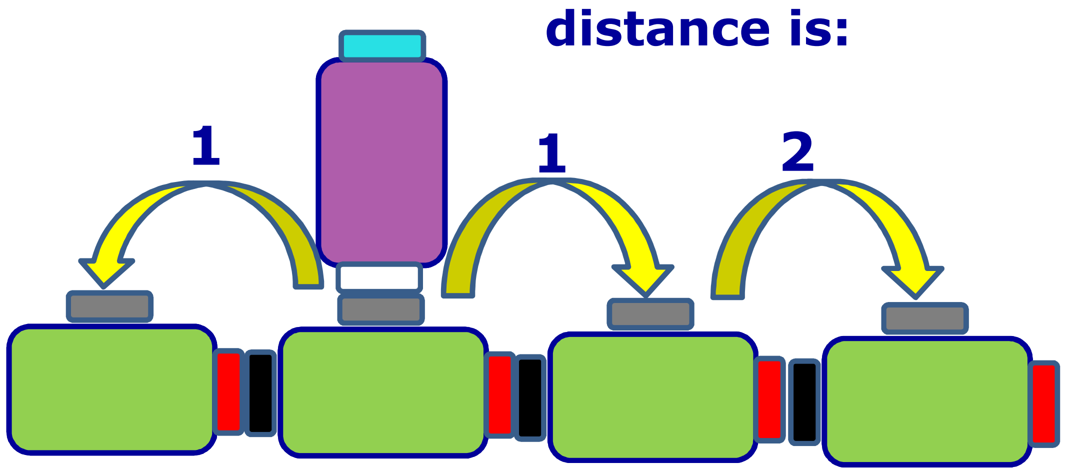
Other elements:



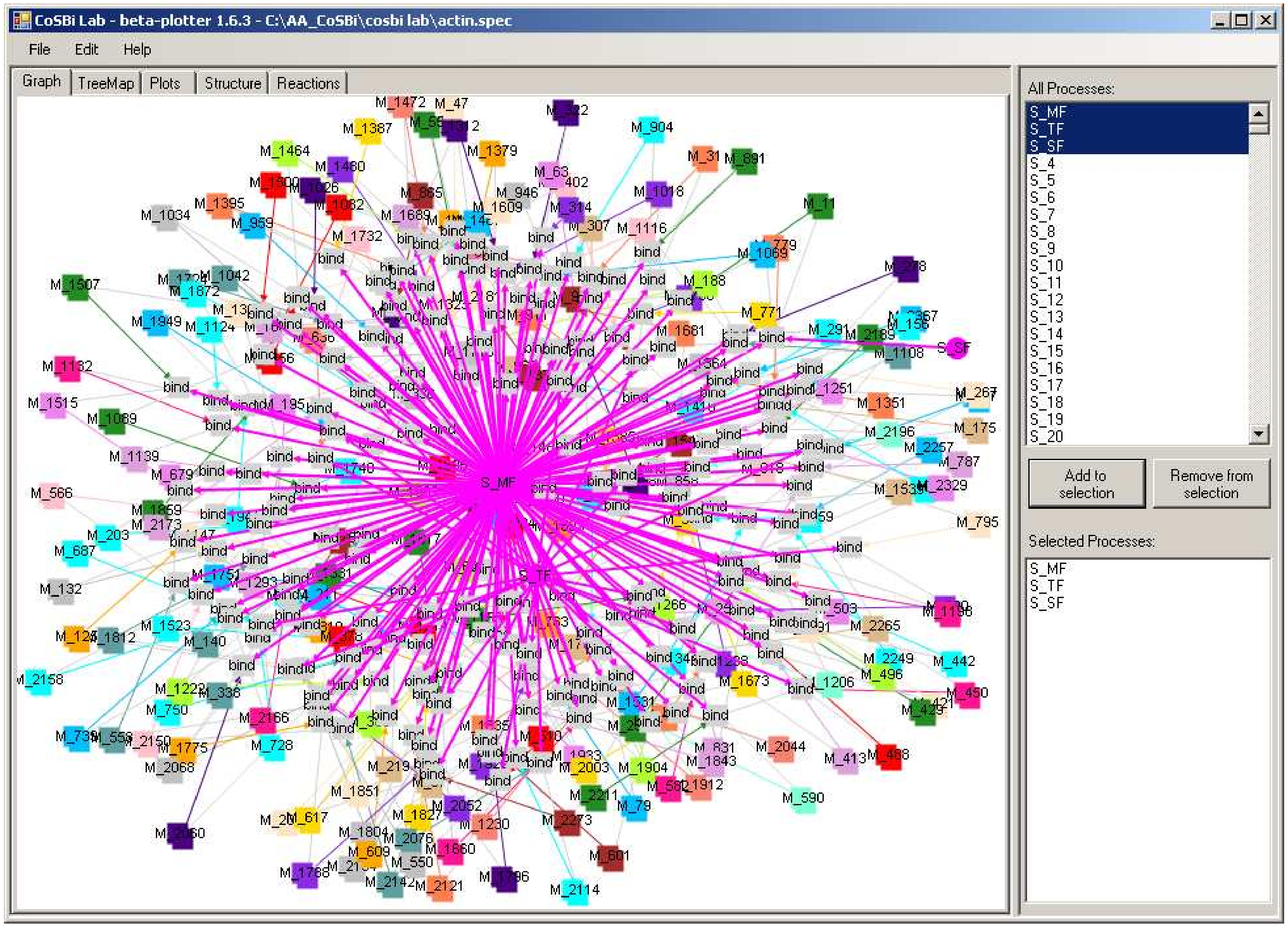


Actin polymerization

Protocol to control proximity of branches



Polymerization computationally



All Processes: S_MF, S_TF, S_SF, S_4, S_5, S_6, S_7, S_8, S_9, S_10, S_11, S_12, S_13, S_14, S_15, S_16, S_17, S_18, S_19, S_20

Selected Processes: S_MF, S_TF, S_SF

Thanks!