Bio-PEPA Model Report

Bio-PEPA Workbench

April 24, 2009

1 Bio-PEPA model

$$
\begin{aligned}
r_1 &= [k_1 \times E \times S] \\
r_{-1} &= [k_{-1} \times E{:}S] \\
r_2 &= [k_2 \times E{:}S] \\
E &= r_1{\downarrow} + r_{-1}{\uparrow} + r_2{\uparrow} \\
S &= r_1{\downarrow} + r_{-1}{\uparrow} \\
E{:}S &= r_1{\uparrow} + r_{-1}{\downarrow} + r_2{\downarrow} \\
P &= r_2{\uparrow} \\
(E \underset{\{r_1, r_{-1}, r_2\}}{\bowtie} (S \underset{\{r_1, r_{-1}\}}{\bowtie} (E{:}S \underset{\{r_2\}}{\bowtie} P)))
\end{aligned}
$$

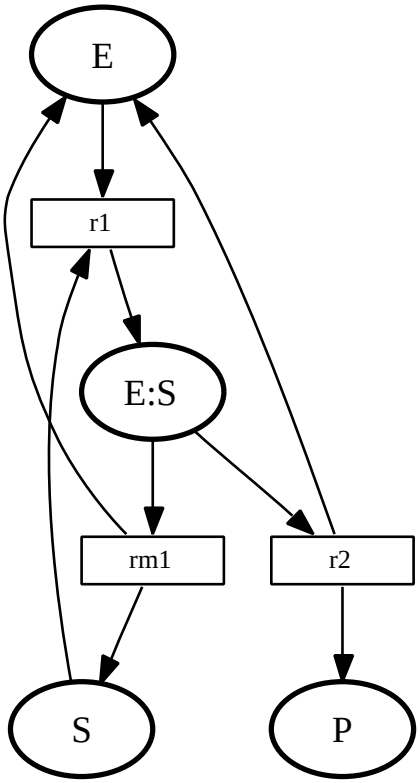
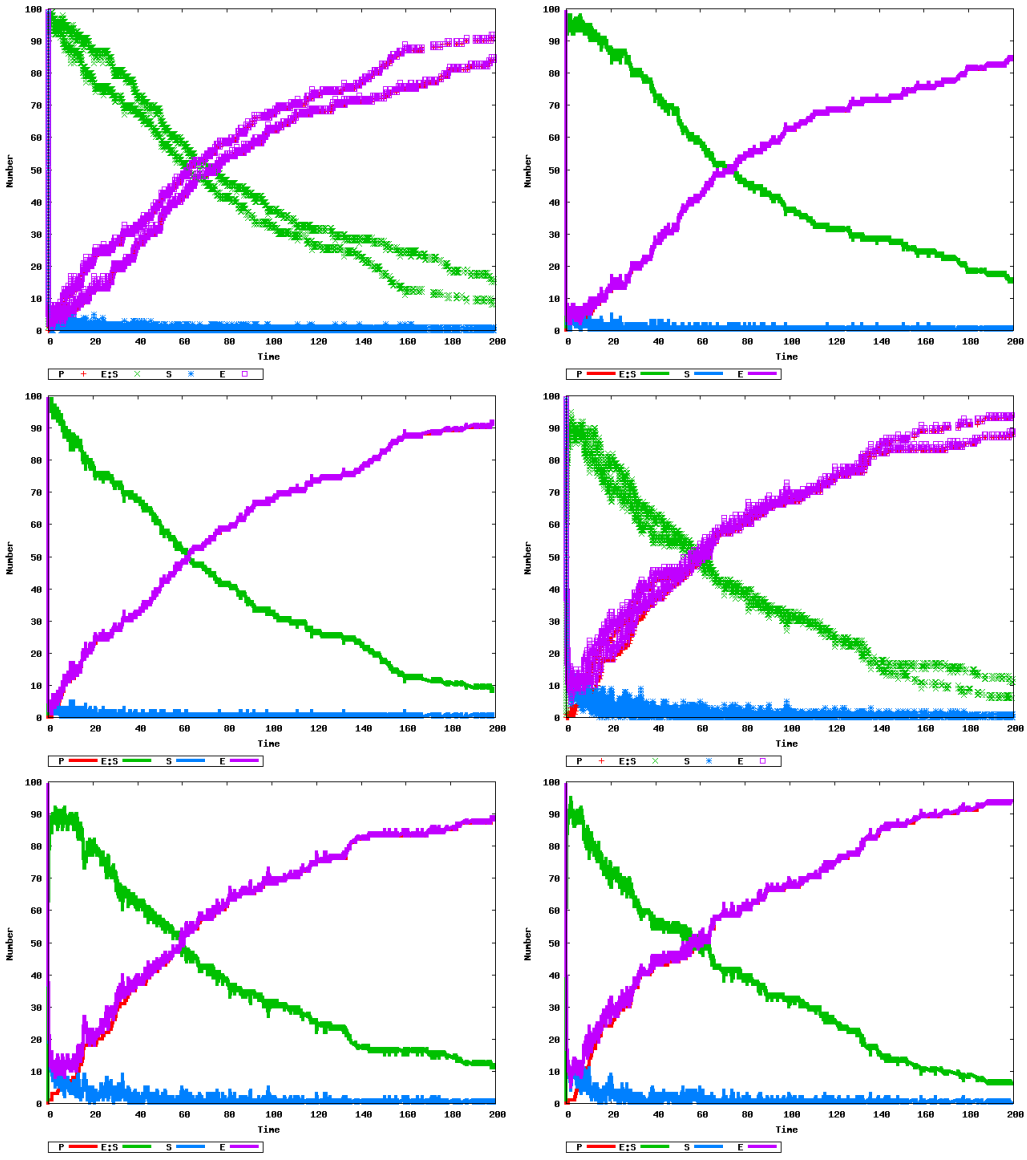
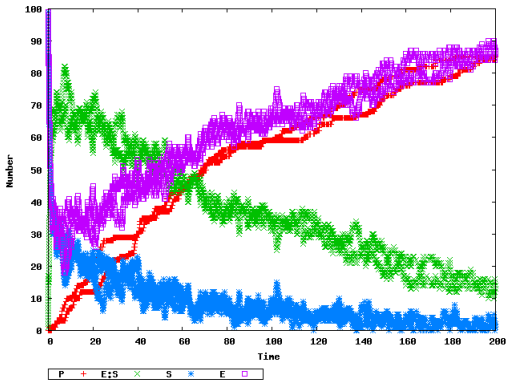
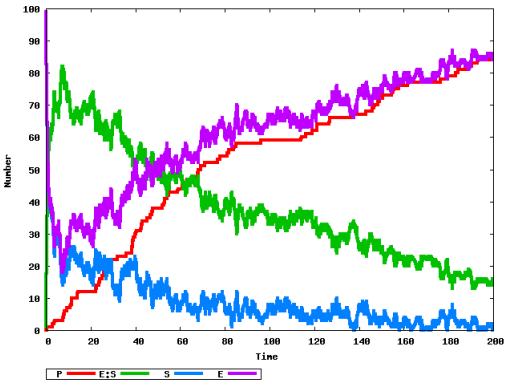


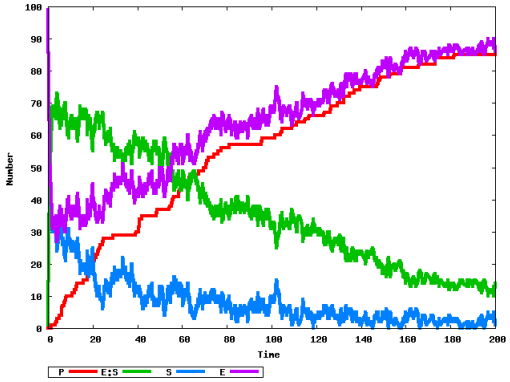
Fig. 1. Reaction network

2 Graphs from Bio-PEPA model




Number
Time
P
E:S
S
E


Number
Time
P
E:S
S
E


Number
Time
P
E:S
S
E

A Bio-PEPA input file

```
% Bio-PEPA model of Michaelis-Menten kinetics

% Species are altered by reactions with stoichiometry
% coefficients

%biopepa.independent.replications: 2
%biopepa.simulation.stoptime: 200
%biopepa.show.all.replications: true
%stochkit.opt.progress.interval: 1L

r1 = [ k1 * E * S ];
rm1 = [ km1 * E:S ];
r2 = [ k2 * E:S ];

E = r1<< +  rm1>>  +  r2>> ;
S = r1<<  +  rm1>> ;
E:S = r1>>  +  rm1<<  +  r2<< ;
P = r2>> ;

(E <r1, rm1, r2> (S <r1, rm1> (E:S <r2> P)))
```

B Dizzy equivalent input file

```
// Dizzy model generated by the BioPEPA Workbench

#model "mm001";

E = 100;
S = 100;
E_colon_S = 0;
P = 0;
k1 = 1;
km1 = 0.1;
k2 = 0.01;

"r1", E + S -> E_colon_S, [ k1 * E * S ] ;
"rm1", E_colon_S -> E + S, [ km1 * E_colon_S ] ;
"r2", E_colon_S -> E + P, [ k2 * E_colon_S ] ;
```

C PRISM equivalent input file

```
// PRISM model compiled from Bio-PEPA input file "mm" by
// Bio-PEPA Workbench Version 1.0 "Charlie Mingus" [22nd-April-2009]

ctmc

const double _k1 = 1;
const double _km1 = 0.1;
const double _k2 = 0.01;

module Rates

  [_r1] (( _k1 * _E * _S ) > 0) -> ( _k1 * _E * _S ) : true;
  [_rm1] (( _km1 * _E_colon_S ) > 0) -> ( _km1 * _E_colon_S ) : true;
  [_r2] (( _k2 * _E_colon_S ) > 0) -> ( _k2 * _E_colon_S ) : true;

endmodule

// Species: _E, _S, _E_colon_S, _P

const int MAX = 100 + 100 + 0 + 0;

module _E

  _E : [0..MAX] init 100;

  [_r1] (_E >= 1) -> 1 : (_E' = _E - 1);
  [_rm1] (_E + 1 <= MAX) -> 1 : (_E' = _E + 1);
  [_r2] (_E + 1 <= MAX) -> 1 : (_E' = _E + 1);

endmodule

module _S

  _S : [0..MAX] init 100;

  [_r1] (_S >= 1) -> 1 : (_S' = _S - 1);
  [_rm1] (_S + 1 <= MAX) -> 1 : (_S' = _S + 1);

endmodule

module _E_colon_S

  _E_colon_S : [0..MAX] init 0;
```

```
  [_r1] (_E_colon_S + 1 <= MAX) -> 1 : (_E_colon_S' = _E_colon_S + 1);
  [_rm1] (_E_colon_S >= 1) -> 1 : (_E_colon_S' = _E_colon_S - 1);
  [_r2] (_E_colon_S >= 1) -> 1 : (_E_colon_S' = _E_colon_S - 1);

endmodule

module _P

  _P : [0..MAX] init 0;

  [_r2] (_P + 1 <= MAX) -> 1 : (_P' = _P + 1);

endmodule


// count rewards: "number of occurrences of r1"
rewards "_r1"
  [_r1] true : 1;
endrewards

// count rewards: "number of occurrences of rm1"
rewards "_rm1"
  [_rm1] true : 1;
endrewards

// count rewards: "number of occurrences of r2"
rewards "_r2"
  [_r2] true : 1;
endrewards

// rewards: "number of E molecules present"
rewards "_E"
  true : _E;
endrewards

// rewards: "square of number of E molecules present (used to calculate standard derivation
rewards "_E_squared"
  true : _E * _E;
endrewards

// rewards: "number of S molecules present"
rewards "_S"
  true : _S;
endrewards
```

```
// rewards: "square of number of S molecules present (used to calculate standard deviation)
rewards "_S_squared"
  true : _S * _S;
endrewards

// rewards: "number of E:S molecules present"
rewards "_E_colon_S"
  true : _E_colon_S;
endrewards

// rewards: "square of number of E:S molecules present (used to calculate standard deviatio
rewards "_E_colon_S_squared"
  true : _E_colon_S * _E_colon_S;
endrewards

// rewards: "number of P molecules present"
rewards "_P"
  true : _P;
endrewards

// rewards: "square of number of P molecules present (used to calculate standard deviation)
rewards "_P_squared"
  true : _P * _P;
endrewards



// End PRISM model compiled from mm
```