

# Stochastic Simulation Library

## Assignment for Selected Topics in Programming

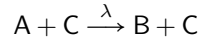
Marius Mikučionis [marius@cs.aau.dk](mailto:marius@cs.aau.dk)

April 24, 2025

### 1 Introduction

A rich class of stochastic systems can be simulated using Doob-Gillespie algorithm. Chemical reactions, biological ecosystems, traffic, and even disease epidemics can be modeled as a stochastic systems. Curiously, such systems exhibit emergent behavior: agents follow simple rules and give rise to a complex system behavior which is not part of any individual rules. Moreover, the system is probabilistic and independent simulations may exhibit different behaviors, therefore statistical methods are needed to understand and predict the effects.

A stochastic system can be described by a number of agents of various species which interact, consume, and produce agents of other species. The interactions are described by a network of reaction rules. For example, the following reaction takes one agent of species A and one agent of C (catalyst) and produce one agent of B with a rate of  $\lambda$ :



The rate  $\lambda$  describes the intrinsic probability of individuals meeting and reacting over one time unit. If any input individuals are missing, then the reaction is not possible. If there are multiple individuals available, then the probability for such reaction is proportionally greater, therefore the overall reaction rate is *multiplied* by the number of input individuals available.

Figure 1 shows an evolution of the system described by the above equation, where the reaction rate is twice as high when there are two catalysing individuals C (the amount A is halved twice as fast), and the reaction has a similar half-life (period until the quantity is halved) even when starting with half the amounts. Observe that the quantities follow an [exponential decay](#) characteristic to the reaction rate being proportional to the input quantities.

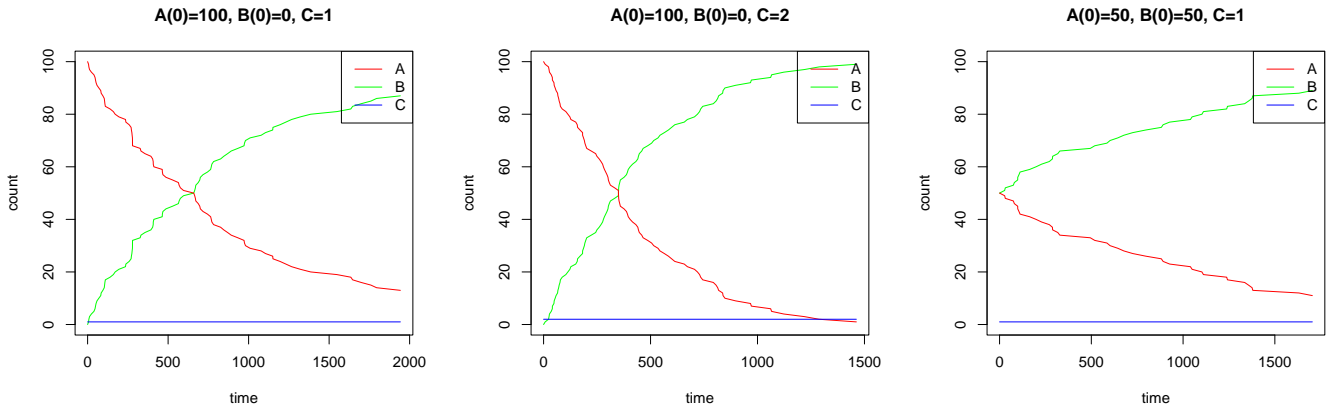


Figure 1: Sample trajectories of A, B and C quantities over time given  $\lambda = 0.001$ .

The reaction delay (time between reaction events) is stochastic: it follows an exponential distribution with the specific (exponential) reaction rate. If there are multiple reactions possible at the same time, then the reaction with the shortest delay is taken next in the simulation.

The goal of the assignment is to develop a library for stochastic simulations described as reaction networks.

## 1.1 Stochastic Simulation of Reactions

A reaction can be simulated as a sequence of reaction steps. In each step small amounts of reactants react in necessary proportions to produce the products of the reaction. For example, when hard groundwater is exposed to air, carbon dioxide dissolves into carbonate ions which meet calcium ions to form milky [limescale](#) precipitate:  $\text{CO}_3^{2-} + \text{Ca}^{2+} \xrightarrow{\lambda} \text{CaCO}_3$ . If  $Q_{\text{CO}_3^{2-}}$  is the number of carbonate ions,  $Q_{\text{Ca}^{2+}}$  is the number of calcium ions, and  $Q_{\text{CaCO}_3}$  is the number of calcium carbonate molecules, then the reaction step is simulated by subtracting 1 ion from  $Q_{\text{CO}_3^{2-}}$ , 1 from  $Q_{\text{Ca}^{2+}}$  and adding 1 to  $Q_{\text{CaCO}_3}$ . When input amounts decrease, reactants are less likely to meet and hence takes longer to react. The time to reaction is stochastic and is generated as follows:

$$r.\text{delay} = \text{Exp}\left(r \cdot \lambda \cdot \prod_{i \in r.\text{inputs}} Q_i\right) \quad (1)$$

Here,  $\text{Exp}(\lambda)$  is a random number distributed according to the [exponential distribution](#) with the rate parameter  $\lambda$  (see [std::exponential\\_distribution](#)),  $Q_i$  is the current amount of the reactant  $i$ . After *delay* time units, the reaction step is performed by consuming the input reactants ( $Q_i := Q_i - 1$ ) and producing the reaction products ( $Q_i := Q_i + 1$ ) (lines 8–9 in Alg. 1), as well as computing the new *delay* (line 3). If some input reactant is exhausted, then such reaction is not possible (its delay is infinite).

Algorithm 1 shows how several reactions are simulated. The algorithm uses reactions  $R$  and maintains the state of the system as the amounts of all the reactants  $\langle Q_i \rangle$ . It always picks a reaction with the smallest delay (line 4). An input of one reaction can be a product of another and vice versa, thus forming reaction chains. The delays of *all* reactions are recomputed (line 3) even though only one reaction is executed at a time. All delays can be recomputed at each iteration because the exponential distribution is memory-less. A more efficient implementation could recompute only the delays of the reactions whose rates were affected by the change in quantities. Another improvement could be to treat catalyst agents (present in both inputs and products) specially by omitting the decrement and the increment of their quantities.

**Input** : Set of reactions  $R$ , end time  $T$ , State  $\langle Q_i \rangle$  with amounts  $Q_i$  of reactant  $i$ .

```

1  $t := 0$ ;
2 while  $t \leq T$  do
3   foreach  $r \in R$  do Compute  $r.\text{delay}$ ;
4    $r := \text{argmin}_{r \in R} r.\text{delay}$ ;
5    $t := t + r.\text{delay}$ ;
6   if  $\forall i \in r.\text{inputs} : Q_i > 0$ 
7     then
8       foreach  $i \in r.\text{input}$  do  $Q_i := Q_i - 1$ ;
9       foreach  $i \in r.\text{product}$  do  $Q_i := Q_i + 1$ ;
10    end
11    Print/store/observe the state  $\langle Q_i \rangle$ ;
12 end
```

**Algorithm 1:** Simulation of multiple reactions

## 2 Requirements

The goal of the assignment is to develop a library for creating and simulating reactions. The usage of the library should be demonstrated on three example stochastic systems, sample programs and unit tests. The solution should contain the following features:

1. The library should *overload operators* to support the reaction rule typesetting directly in C++ code.
2. Provide *pretty-printing* of the reaction network in a) human readable format and b) network graph (e.g. Fig. 4).
3. Implement a [generic symbol table](#) to *store* and *lookup* objects of user-defined key and value types. Support failure cases when a) the table does not contain a looked up symbol, b) the table already contains a symbol that is being added. Demonstrate the usage of the symbol table with the reactants (names and initial counts).
4. Implement the *stochastic simulation* (Alg. 1) of the system using the reaction rules.
5. Demonstrate the *application* of the library on the three examples (shown in Fig. 1, 2, 3).
6. *Display* simulation trajectories of how the amounts change. External tools/libraries can be used to visualize.
7. Implement a [generic support](#) for (any) user-supplied state observer function object *or* provide a lazy trajectory generation interface (coroutine). The observer itself should be part by the user/test program and **not** part of the library. To demonstrate the generic support, estimate the peak of hospitalized agents in Covid-19 example *without storing an entire trajectory*. Record the peak hospitalization values for population sizes of  $N_{\text{NJ}}$  and  $N_{\text{DK}}$ .
8. Implement support for *multiple CPU cores by parallelizing* the computation of several simulations at the same time. Estimate the likely (average) value of the hospitalized peak over 100 simulations.
9. Implement *unit tests* (e.g. test symbol table methods, their failure cases, pretty-printing reaction rules, etc).
10. *Benchmark* and compare the stochastic simulation performance (e.g. the time it takes to compute 100 simulations a single core, multiple cores, *or* improved implementation). Record the timings and make your conclusions.

**Make sure to include** testing and benchmarking code as well as the *sample results, measurements, plots and conclusions* into the report.

## A Example Library-Usage Code

The code are just suggested examples, one may choose a different design, provided that the requirements are fulfilled.

### A.1 Genetic Oscillator for Circadian Rhythm

Table 1 shows [the circadian rhythm reactions in humans](#), C++ code in Listing 1 and a trajectory in Fig. 2.

Table 1: Reactions and values for circadian rhythm:  $\emptyset$  decay into environment,  $\text{m}^{-1}$  – “per molecule”,  $\text{h}^{-1}$  – “per hour”.

$A + D_A \xrightarrow{\gamma_A} D'_A$	$M_A \xrightarrow{\beta_A} M_A + A$	$\alpha_A = 50 \text{ h}^{-1}$	$\delta_{M_R} = 0.5 \text{ h}^{-1}$
$D'_A \xrightarrow{\theta_A} D_A + A$	$M_R \xrightarrow{\beta_R} M_R + R$	$\alpha'_A = 500 \text{ h}^{-1}$	$\theta_A = 50 \text{ h}^{-1}$
$A + D_R \xrightarrow{\gamma_R} D'_R$	$A + R \xrightarrow{\gamma_C} C$	$\alpha_R = 0.01 \text{ h}^{-1}$	$\theta_R = 100 \text{ h}^{-1}$
$D'_R \xrightarrow{\theta_R} D_R + A$	$C \xrightarrow{\delta_A} R$	$\alpha'_R = 50 \text{ h}^{-1}$	$D_A = 1 \text{ m}$
$D'_A \xrightarrow{\alpha'_A} M_A + D'_A$	$A \xrightarrow{\delta_A} \emptyset$	$\beta_A = 50 \text{ h}^{-1}$	$D'_A = 0$
$D_A \xrightarrow{\alpha_A} M_A + D_A$	$R \xrightarrow{\delta_R} \emptyset$	$\beta_R = 5 \text{ h}^{-1}$	$D_R = 1 \text{ m}$
$D'_R \xrightarrow{\alpha'_R} M_R + D'_R$	$M_A \xrightarrow{\delta_{M_A}} \emptyset$	$\gamma_A = 1 \text{ m}^{-1} \text{ h}^{-1}$	$D'_R = 0$
$D_R \xrightarrow{\alpha_R} M_R + D_R$	$M_R \xrightarrow{\delta_{M_R}} \emptyset$	$\gamma_R = 1 \text{ m}^{-1} \text{ h}^{-1}$	$M_A = 0$
		$\gamma_C = 2 \text{ m}^{-1} \text{ h}^{-1}$	$M_R = 0$
		$\delta_A = 1 \text{ h}^{-1}$	$A = 0$
		$\delta_R = 0.2 \text{ h}^{-1}$	$R = 0$
		$\delta_{M_A} = 10 \text{ h}^{-1}$	$C = 0$

Listing 1: Circadian oscillator in C++

```

1 Vessel circadian_rhythm()
2 {
3     const auto alphaA = 50;
4     const auto alpha_A = 500;
5     const auto alphaR = 0.01;
6     const auto alpha_R = 50;
7     const auto betaA = 50;
8     const auto betaR = 5;
9     const auto gammaA = 1;
10    const auto gammaR = 1;
11    const auto gammaC = 2;
12    const auto deltaA = 1;
13    const auto deltaR = 0.2;
14    const auto deltaMA = 10;
15    const auto deltaMR = 0.5;
16    const auto thetaA = 50;
17    const auto thetaR = 100;
18    auto v = stochastic::Vessel{"Circadian
    ↪Rhythm"};
19    const auto env = v.environment();
20    const auto DA = v.add("DA", 1);
21    const auto D_A = v.add("D_A", 0);
22    const auto DR = v.add("DR", 1);
23
24    const auto D_R = v.add("D_R", 0);
25    const auto MA = v.add("MA", 0);
26    const auto MR = v.add("MR", 0);
27    const auto A = v.add("A", 0);
28    const auto R = v.add("R", 0);
29    const auto C = v.add("C", 0);
30    v.add((A + DA) >> gammaA >>= D_A);
31    v.add(D_A >> thetaA >>= DA + A);
32    v.add((A + DR) >> gammaR >>= D_R);
33    v.add(D_R >> thetaR >>= DR + A);
34    v.add(D_A >> alpha_A >>= MA + D_A);
35    v.add(D_R >> alpha_R >>= MR + D_R);
36    v.add(DR >> alphaR >>= MR + DR);
37    v.add(MA >> betaA >>= MA + A);
38    v.add(MR >> betaR >>= MR + R);
39    v.add((A + R) >> gammaC >>= C);
40    v.add(C >> deltaA >>= R);
41    v.add(A >> deltaA >>= env);
42    v.add(R >> deltaR >>= env);
43    v.add(MA >> deltaMA >>= env);
44    v.add(MR >> deltaMR >>= env);
45    return v;
46 }

```

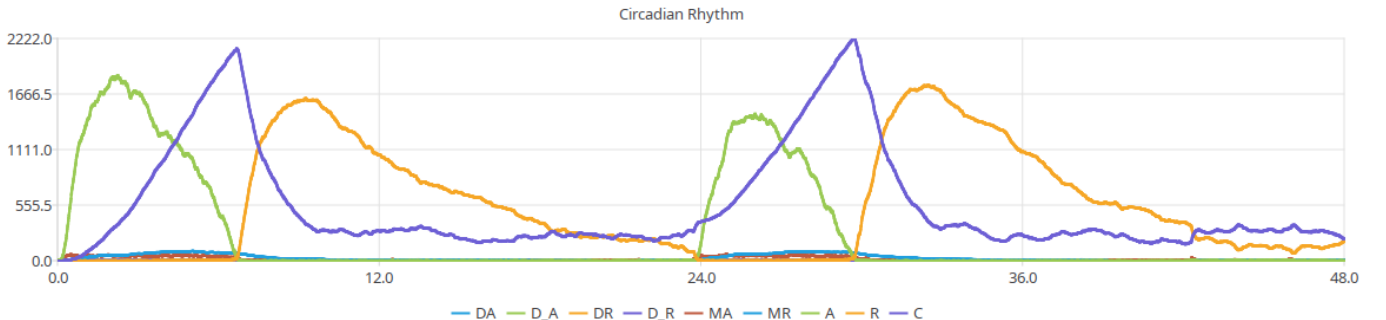


Figure 2: Sample trajectory of circadian rhythm: A, C, R repeat with  $\approx 24$  hour periods (using QtCharts).

## A.2 SEIHR Model for Covid-19

Table 2 describes Covid-19 spread model (without quarantine), C++ code in Listing 2 and a trajectory in Fig. 3.

Table 2: Reactions and values for Covid-19 model, where  $d^{-1}$  means “per day”.

$S + I \xrightarrow{\beta/N} E + I$	$N = 10^4$	$R_0 = 2.4$
$E \xrightarrow{\alpha} I$	$\varepsilon = 9 \cdot 10^{-4}$	$\alpha = 1/5.1 d^{-1}$
$I \xrightarrow{\gamma} R$	$E = N \cdot \varepsilon \cdot 15$	$\gamma = 1/3.1 d^{-1}$
$I \xrightarrow{\kappa} H$	$I = N \cdot \varepsilon$	$\beta = R_0 \cdot \gamma$
$H \xrightarrow{\tau} R$	$H = 0$	$P_H = 9 \cdot 10^{-4}$
	$R = 0$	$\kappa = \gamma \cdot P_H \cdot (1 - P_H)$
	$N_{DK} = 5'822'763$	$\tau = 1/10.12 d^{-1}$
	$N_{NJ} = 589'755$	

Listing 2: Covid-19 model in C++

```

1  /// SEIHR model for COVID19 epidemic with population size N
2  Vessel seihr(uint32_t N)
3  {
4      auto v = Vessel{"COVID19 SEIHR: " + std::to_string(N)};
5      const auto eps = 0.0009;           // initial fraction of infectious
6      const auto I0 = size_t(std::round(eps * N)); // initial infectious
7      const auto E0 = size_t(std::round(eps * N * 15)); // initial exposed
8      const auto S0 = N - I0 - E0;       // initial susceptible
9      const auto R0 = 2.4;               // initial basic reproductive number
10     const auto alpha = 1.0 / 5.1;       // incubation rate (E -> I) ~5.1 days
11     const auto gamma = 1.0 / 3.1;       // recovery rate (I -> R) ~3.1 days
12     const auto beta = R0 * gamma;       // infection/generation rate (S+I -> E+I)
13     const auto P_H = 0.9e-3;            // probability of hospitalization
14     const auto kappa = gamma * P_H * (1.0 - P_H); // hospitalization rate (I -> H)
15     const auto tau = 1.0 / 10.12;       // removal rate in hospital (H -> R)
16     const auto S = v.add("S", S0);      // susceptible
17     const auto E = v.add("E", E0);      // exposed
18     const auto I = v.add("I", I0);      // infectious
19     const auto H = v.add("H", 0);        // hospitalized
20     const auto R = v.add("R", 0);        // removed/immune (recovered + dead)
21     v.add((S + I) >> beta / N >=> E + I); // susceptible becomes exposed by infectious
22     v.add(E >> alpha >=> I);           // exposed becomes infectious
23     v.add(I >> gamma >=> R);           // infectious becomes removed
24     v.add(I >> kappa >=> H);           // infectious becomes hospitalized
25     v.add(H >> tau >=> R);             // hospitalized becomes removed
26     return v;
27 }

```

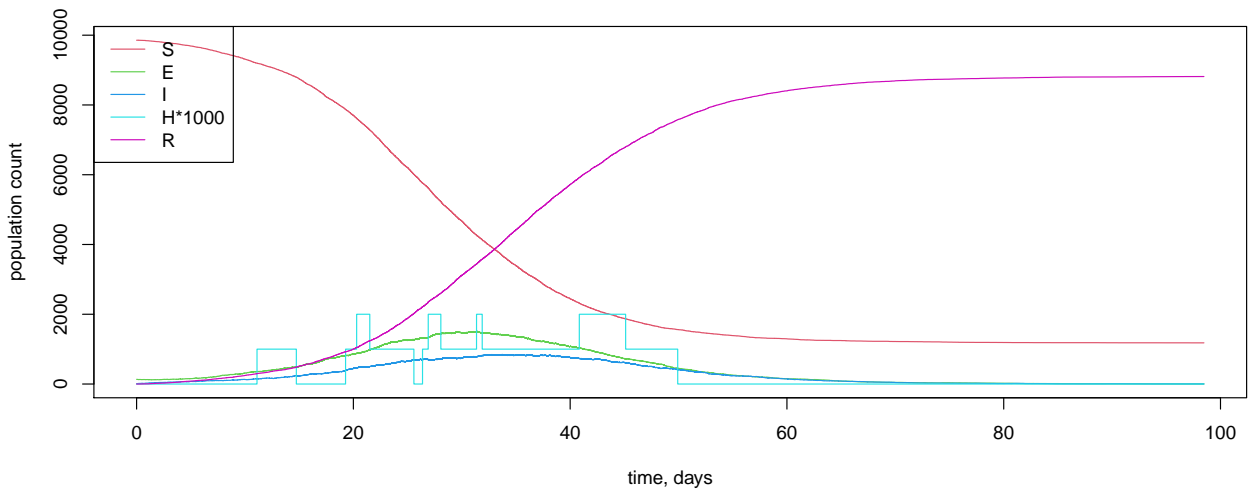


Figure 3: Sample trajectory of Covid-19 population  $N = 10000$ : hospitalizations peak at 2 (using [gnuplot](#)).

## B Example Graph Layouts

This particular visualization is just an example and not part of the requirements: one may choose a different graph visualization framework and render the image using other *tools*.

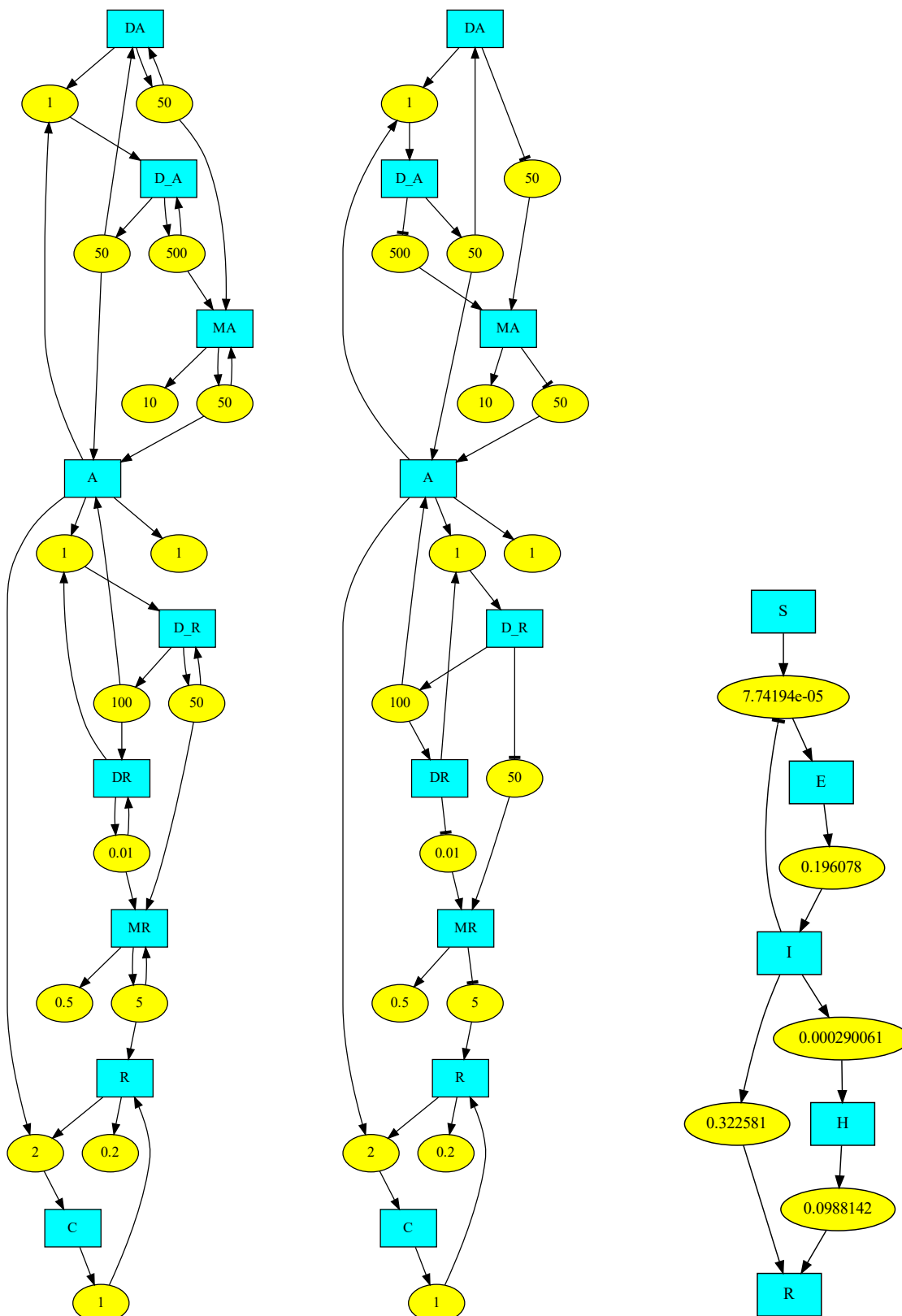


Figure 4: Layout of reaction networks: circadian model, circadian with catalysts, Covid-19 from Listings 3,4,5. The images are rendered using command-line `dot` utility from [Graphviz](#).

## C Example Graph Description

The listings below describe the reaction networks in [Graphviz](#) dot format. The description format is just an example and not part of the requirements: one may choose a different graph visualization framework.

Listing 3: Covid-19 reaction network in dot-format.

```
digraph {
  s0[label="S",shape="box",style="filled",fillcolor="cyan"];
  s1[label="E",shape="box",style="filled",fillcolor="cyan"];
  s2[label="I",shape="box",style="filled",fillcolor="cyan"];
  s3[label="R",shape="box",style="filled",fillcolor="cyan"];
  s4[label="H",shape="box",style="filled",fillcolor="cyan"];
  r0[label="7.74194e-05",shape="oval",style="filled",fillcolor="yellow"];
  s2 -> r0 [arrowhead="tee"];
  s0 -> r0;
  r0 -> s1;
  r1[label="0.196078",shape="oval",style="filled",fillcolor="yellow"];
  s1 -> r1;
  r1 -> s2;
  r2[label="0.322581",shape="oval",style="filled",fillcolor="yellow"];
  s2 -> r2;
  r2 -> s3;
  r3[label="0.000290061",shape="oval",style="filled",fillcolor="yellow"];
  s2 -> r3;
  r3 -> s4;
  r4[label="0.0988142",shape="oval",style="filled",fillcolor="yellow"];
  s4 -> r4;
  r4 -> s3;
}
```

Listing 4: Circadian rhythm reaction network in dot-format.

```
digraph {
  s0[label="DA",shape="box",style="filled",fillcolor="cyan"];
  s1[label="D_A",shape="box",style="filled",fillcolor="cyan"];
  s2[label="DR",shape="box",style="filled",fillcolor="cyan"];
  s3[label="D_R",shape="box",style="filled",fillcolor="cyan"];
  s4[label="MA",shape="box",style="filled",fillcolor="cyan"];
  s5[label="MR",shape="box",style="filled",fillcolor="cyan"];
  s6[label="A",shape="box",style="filled",fillcolor="cyan"];
  s7[label="R",shape="box",style="filled",fillcolor="cyan"];
  s8[label="C",shape="box",style="filled",fillcolor="cyan"];
  r0[label="1",shape="oval",style="filled",fillcolor="yellow"];
  s6 -> r0;
  s0 -> r0;
  r0 -> s1;
  r1[label="50",shape="oval",style="filled",fillcolor="yellow"];
  s1 -> r1;
  r1 -> s0;
  r1 -> s6;
  r2[label="1",shape="oval",style="filled",fillcolor="yellow"];
  s6 -> r2;
  s2 -> r2;
  r2 -> s3;
  r3[label="100",shape="oval",style="filled",fillcolor="yellow"];
  s3 -> r3;
  r3 -> s2;
  r3 -> s6;
  r4[label="500",shape="oval",style="filled",fillcolor="yellow"];
  s1 -> r4;
  r4 -> s4;
  r4 -> s1;
  r5[label="50",shape="oval",style="filled",fillcolor="yellow"];
  s0 -> r5;
  r5 -> s4;
  r5 -> s0;
  r6[label="50",shape="oval",style="filled",fillcolor="yellow"];
  s3 -> r6;
  r6 -> s5;
  r6 -> s3;
  r7[label="0.01",shape="oval",style="filled",fillcolor="yellow"];
  s2 -> r7;
  r7 -> s5;
  r7 -> s2;
  r8[label="50",shape="oval",style="filled",fillcolor="yellow"];
  s4 -> r8;
  r8 -> s4;
  r8 -> s6;
  r9[label="5",shape="oval",style="filled",fillcolor="yellow"];
  s5 -> r9;
  r9 -> s5;
  r9 -> s7;
  r10[label="2",shape="oval",style="filled",fillcolor="yellow"];
  s6 -> r10;
  s7 -> r10;
  r10 -> s8;
  r11[label="1",shape="oval",style="filled",fillcolor="yellow"];
  s8 -> r11;
  r11 -> s7;
  r12[label="1",shape="oval",style="filled",fillcolor="yellow"];
  s6 -> r12;
  r13[label="0.2",shape="oval",style="filled",fillcolor="yellow"];
  s7 -> r13;
  r14[label="10",shape="oval",style="filled",fillcolor="yellow"];
  s4 -> r14;
  r15[label="0.5",shape="oval",style="filled",fillcolor="yellow"];
  s5 -> r15;
}
```

Listing 5: Circadian rhythm with catalysts reaction network in dot-format.

```

digraph {
  s0[label="DA",shape="box",style="filled",fillcolor="cyan"];
  s1[label="D_A",shape="box",style="filled",fillcolor="cyan"];
  s2[label="DR",shape="box",style="filled",fillcolor="cyan"];
  s3[label="D_R",shape="box",style="filled",fillcolor="cyan"];
  s4[label="MA",shape="box",style="filled",fillcolor="cyan"];
  s5[label="MR",shape="box",style="filled",fillcolor="cyan"];
  s6[label="A",shape="box",style="filled",fillcolor="cyan"];
  s7[label="R",shape="box",style="filled",fillcolor="cyan"];
  s8[label="C",shape="box",style="filled",fillcolor="cyan"];
  r0[label="I",shape="oval",style="filled",fillcolor="yellow"];
  s6 -> r0;
  s0 -> r0;
  r0 -> s1;
  r1[label="50",shape="oval",style="filled",fillcolor="yellow"];
  s1 -> r1;
  r1 -> s0;
  r1 -> s6;
  r2[label="1",shape="oval",style="filled",fillcolor="yellow"];
  s2 -> r2;
  s6 -> r2;
  r2 -> s3;
  r3[label="100",shape="oval",style="filled",fillcolor="yellow"];
  s3 -> r3;
  r3 -> s2;
  r3 -> s6;
  r4[label="500",shape="oval",style="filled",fillcolor="yellow"];
  s4 -> r4 [arrowhead="tee"];
  r4 -> s4;

  r5[label="50",shape="oval",style="filled",fillcolor="yellow"];
  s0 -> r5 [arrowhead="tee"];
  r5 -> s4;
  r6[label="50",shape="oval",style="filled",fillcolor="yellow"];
  s3 -> r6 [arrowhead="tee"];
  r6 -> s5;
  r7[label="0.01",shape="oval",style="filled",fillcolor="yellow"];
  s2 -> r7 [arrowhead="tee"];
  r7 -> s5;
  r8[label="50",shape="oval",style="filled",fillcolor="yellow"];
  s4 -> r8 [arrowhead="tee"];
  r8 -> s6;
  r9[label="5",shape="oval",style="filled",fillcolor="yellow"];
  s5 -> r9 [arrowhead="tee"];
  r9 -> s7;
  r10[label="2",shape="oval",style="filled",fillcolor="yellow"];
  s6 -> r10;
  r7 -> r10;
  r10 -> s8;
  r11[label="1",shape="oval",style="filled",fillcolor="yellow"];
  s8 -> r11;
  r11 -> s7;
  r12[label="1",shape="oval",style="filled",fillcolor="yellow"];
  s6 -> r12;
  r13[label="0.2",shape="oval",style="filled",fillcolor="yellow"];
  s7 -> r13;
  r14[label="10",shape="oval",style="filled",fillcolor="yellow"];
  s4 -> r14;
  r15[label="0.5",shape="oval",style="filled",fillcolor="yellow"];
  s5 -> r15;
}

```

Listing 6: Listing style code for typesetting dot format in L<sup>A</sup>T<sub>E</sub>X.

```

\lstdefinlanguage{dot}{
  keywords={digraph, label, shape, style, fillcolor},
  sensitive=true, string=[d]{"}
}

\lstdefinestyle{colorDot}{
  language=dot, basicstyle=\ttfamily\scriptsize,
  keywordstyle=\textcolor{blue},
  stringstyle=\slshape\textcolor{red!70!black},
  commentstyle=\slshape\textcolor{green!50!black},
  morecomment=[s][\bfseries\slshape\textcolor{green!50!black}]{/**}{*/},
  tabsize=4,
  showstringspaces=false,
  breaklines=true, breakatwhitespace=true,
  prebreak={\hbox{\quad\textcolor{red}{\$}\rhookswarrow$}},
  postbreak={\hbox{\textcolor{red}{\$}\lhookrightarrow$}},
  breakindent={-8pt}, breakautoindent=false, % numbers=left, numberstyle=\tiny,
  frameshape={RYY}{N}{YYY} % frame=tb, frameround=tttt
}

```

## D Semantics of Rules

This section explains formally the meaning of the reaction rules.

For  $k, \ell, m, n$  positive integer counts,  $R_i$  positive integer count of species  $i$  and  $t$  positive real-valued time:

$$\frac{kA + \ell B \xrightarrow{\lambda} mC + nD, \langle Q_i \mid Q_A \geq k \wedge Q_B \geq \ell \rangle}{\text{Prob}(\langle Q_A, Q_B, Q_C, Q_D \rangle \longrightarrow \langle Q_A - k, Q_B - \ell, Q_C + m, Q_D + n \rangle, t) = \text{PDF}_{\exp(\lambda \cdot Q_A \cdot Q_B)}(t)}$$

Given

1. the reaction *rule* where  $k$  count of A meet  $\ell$  count of B with a rate of  $\lambda$  and produce  $m$  count of C and  $n$  count of D, and
2. the *system state*  $\langle Q_A, Q_B, Q_C, Q_D \rangle$  with enough input agent counts:  $Q_A \geq k$  and  $Q_B \geq \ell$ ,

Then

- the probability of a transition from state  $\langle Q_A, Q_B, Q_C, Q_D \rangle$  to  $\langle Q_A - k, Q_B - \ell, Q_C + m, Q_D + n \rangle$  after time delay  $t$  is equal to the probability density function of the exponential distribution of rate  $(\lambda \cdot Q_A \cdot Q_B)$  at  $t$ .

### D.1 Simplified Rule with Single Molecules

$$\frac{A + B \xrightarrow{\lambda} C + D, \langle Q_i \mid Q_A > 0 \wedge Q_B > 0 \rangle}{\text{Prob}(\langle Q_A, Q_B, Q_C, Q_D \rangle \longrightarrow \langle Q_A - 1, Q_B - 1, Q_C + 1, Q_D + 1 \rangle, t) = \text{PDF}_{\exp(\lambda \cdot Q_A \cdot Q_B)}(t)}$$