Three-dimensional description of the spontaneous onset of homochirality on the surface of a conglomerate crystal phase. *Supporting information*

Raphaël Plasson, Kouichi Asakura, Dilip K. Kondepudi

1 Description of the Program

1.1 Algorithm

A discrete stochastic description of the system has been chosen[1]. Each elementary growth subunit is supposed attached to one element of a three-dimensional matrix, and no diffusion is supposed to occur (this is actually not a real limitation, as solid compounds can't diffuse, and as all liquid compounds are identical). All compounds that can react at a given time t are listed. They can either be a L subunit (liquid subunit, being on top of an interfacial (I) crystalline subunit) or G compounds (growth layer subunit, waiting to become part of the crystal (C) phase). The probability of each one of these subunits to be transformed in another one is calculated on the basis of the corresponding stochastic kinetic rates, and of their respective environment: the four lateral subunit (which can catalyze and induce crystallizations, red block in figure S1) and the underneath subunit (acting as a fixation site, blue block in figure S1).

The total sum of all stochastic rates of all possible events δp is calculated, and a random number is drawn from the uniformly distributed interval $[0, \delta p[$[2]. This number allows to choose which event will actually happen in the infinitesimal time $\delta t = \frac{1}{\delta p}$.

The interest of such an algorithm is to inherently take into account the statistical fluctuations[3] and to keep the microscopic (molecular) description. On top of that, all stochastic rates are in the same unit (s^{-1}) — as representing directly a probability of reaction rather than a macroscopic kinetic rates — which maker easier the comparison between all kinetic rates (which can't be directly done in a macroscopic description, as k_{01}, k_{02} and k_{20} are first order rates, while k_{11} k_{12} and k_{12} are second order reactions)[4].

1.2 Implementation

The program was written in C++, so that an object-oriented structure can guarantee an easy maintenance and extensibility of the source code, and a clear organization of data and processes. The core program is handled by a batch program written in `Python`[5], in order to have access to high level function for automatic plotting of functions (using `Gnuplot`[6]) and the generation of graphical reports (using `PIL`[7]) for each calculation.

1.2.1 Classes

The program was organized in six compounds:

`Medium` class This class contain the 3-D matrix describing the growing crystal, and its link (via `Layer` and `Sublayer` classes) with possible reactions (described in event class). It allows the initiation of calculation for a given number of elementary reaction.

`Layer` class Description of the growth front, divided by sublayers (each sublayer corresponds to one column of the total growth front).

`Sublayer` class One column of the `Layer` class, divided in a list of `Events`. This class and the preceding one are the link between `Medium` and `Event` class. An initially more general `Layer` class have been splitted into these two `Layer` and `Sublayer` classes for calculation time optimization purpose. They permit to choose from the `Medium` class which possible reaction (one instance of `Reac` class) will occur (see the paragraph 1.2.2 for more details).

[1] J. J. Lukkien, J. P. L. Segers, and P. A. J. Hilbers. *Efficient Monte-Carlo methods for the simulation of catalytic surface reactions* **Phys. Rev. E**, 58(2):2598–2613, 1998.

[2] double precision uniformly distributed random number, calculated by the `drand48` function from standard C library

[3] see section 2 for more informations about statistical fluctuations

[4] see section 3 for the relationship between stochastic and macroscopic kinetic rates

[5] `http://python.org`

[6] `http://www.gnuplot.info`

[7] `http://www.pythonware.com/products/pil/`

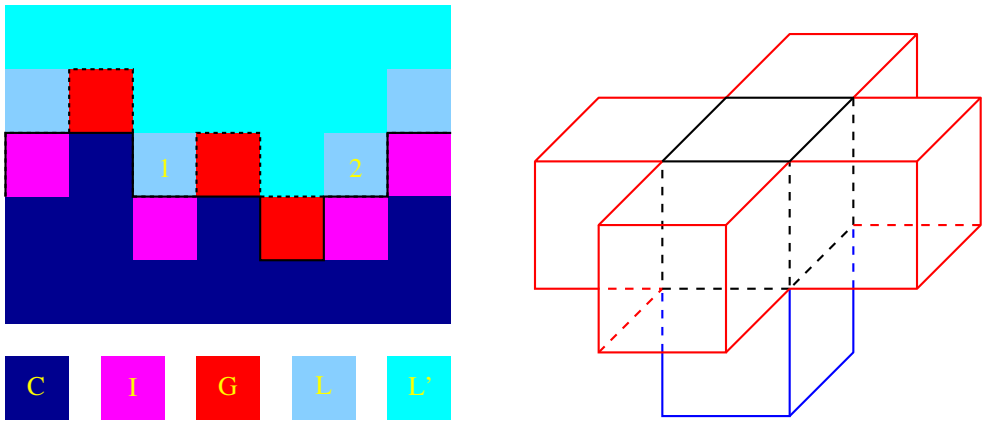

Figure S1: Schematic description of a growing layer. C: crystal bulk phase, I: Interface of crystal to liquid, G: growth layer, L: liquid phase that can react, L': liquid phase that is not in contact with the interface. 3D representation: black, center block; red, lateral block; blue, underneath block.

`Event` class This class instantiate the several possible reactions than can occur in the system, and can perform them when asked through the `Medium->Layer->Sublayer` chain.

`Reac` structure This structure gives all the informations relative to a given reaction (coordinates, kinetic rates, type of reaction, and the kind of compound to be generated)

`Linked_node` class Intermediate class only used by `Medium` for the 3D size distribution determination.

The schematic signification of each class is represented in figure S2. The detail of these classes is given in the UML diagram given in figure S3. The call tree of the main functions is given in figure S4.

1.2.2 Optimizations and performances

A first version of the program only used a `Layer` class, as a direct two dimensional set of events. A profiling of this program has shown that the process was most of the time being calculating which event was to be performed. As the number of event is proportional to the square of the length of the basis (`smax`), the time of calculation was roughly proportional to $\texttt{smax}^2$, i.e. it very quickly increased with the size of the simulated crystal. Practically, very long time of calculation were necessary for sufficiently large systems.

An intermediate class `Sublayer` was thus added between `Layer` and `Event`. It allows to stock in memory the intermediate total of stochastic rates of each sublayer. The localization of the appropriate reaction is then only proportional to `2*smax`. The profiling of this version shows that their are no more deadlocks, and that the process activity is distributed in the several calculation sections (see figure S4).

A typical simulation can be performed — on a `Pentium M` processor, `1.8 GHz`, `1 GB DDRAM` — from few minutes (in the cases of a $200 \times 200 \times 40$ crystal), to few hours (in the case of a $300 \times 300 \times 8192$ crystal). This corresponds to about 100,000 to 150,000 elementary reaction per second, an elementary reaction being computed in 7 to 8 µs.

1.2.3 Reports

The python batch program allows the automation of the calculation of several crystallization systems for different parameters. At the end of each simulation,

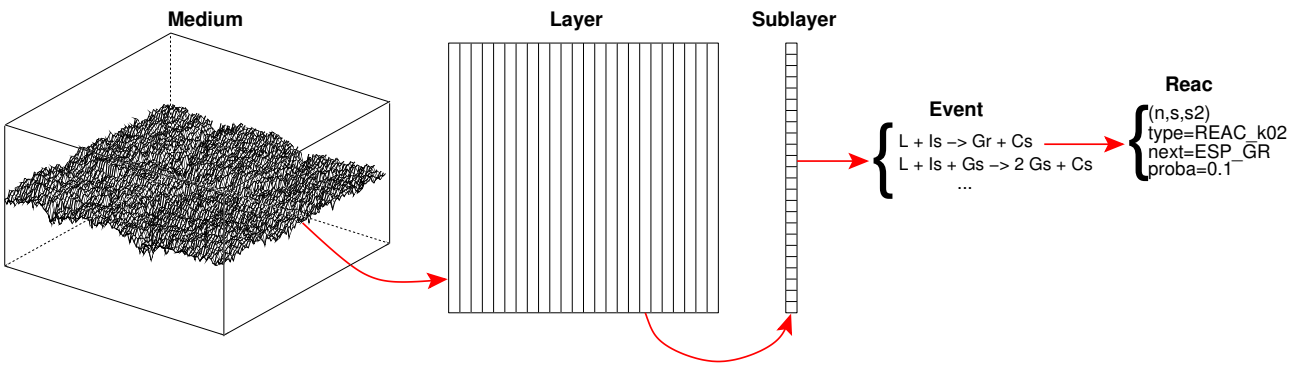

Figure S2: Schematic signification of the program classes.

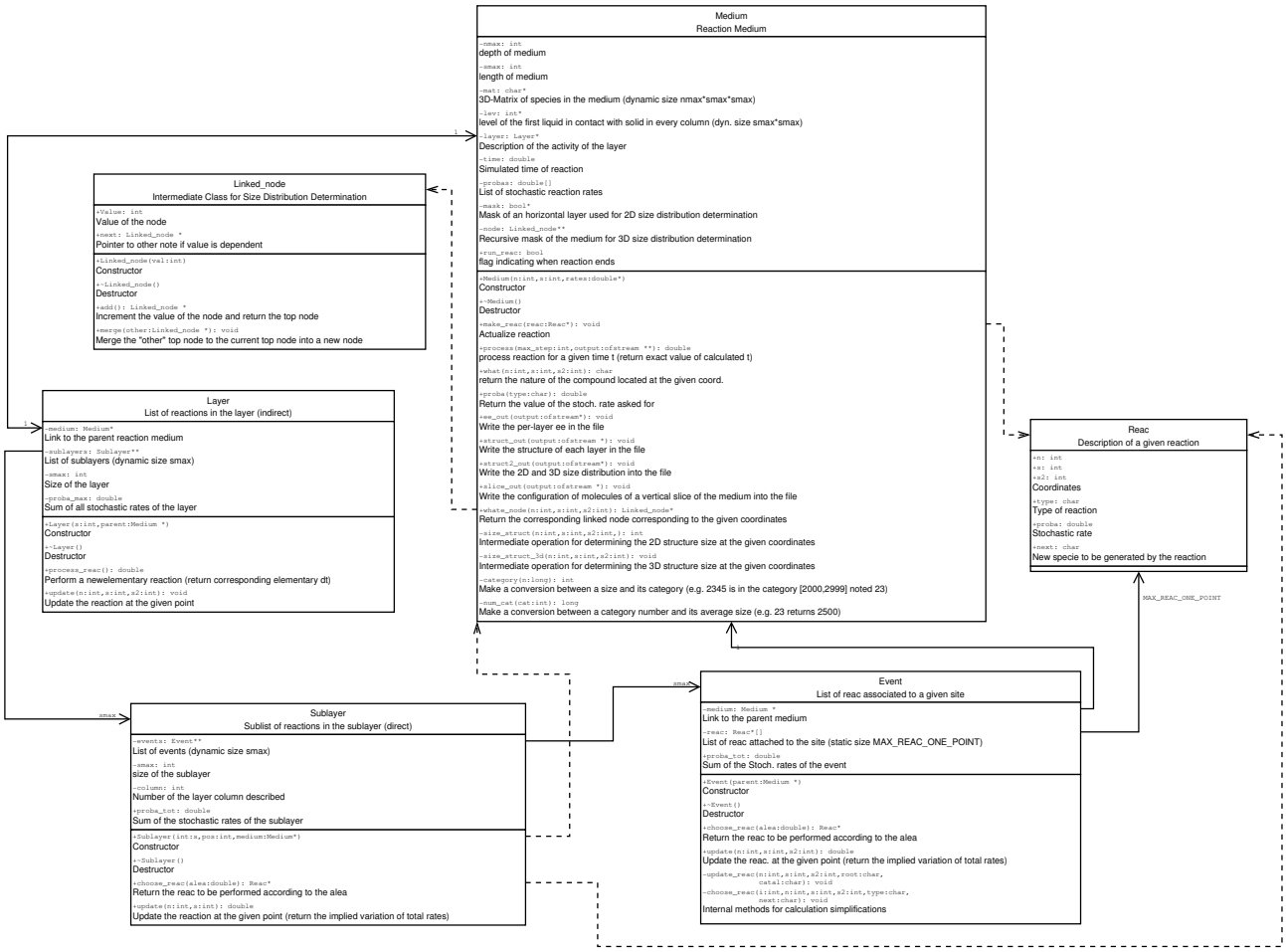

Figure S3: UML Diagram

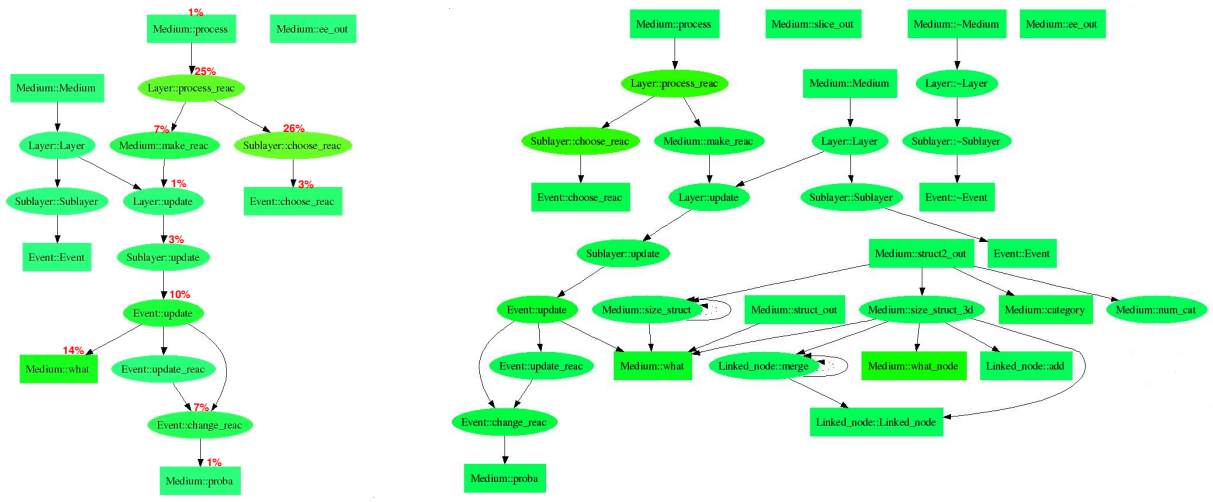

Figure S4: Call tree traced by the GNU profiling tool `oprofile`. In red are indicated the fraction of time spent by the process in each indicated method (only percentage superior to 1% and most used methods are given). The first graph is for a run without structure determination, and the second one includes it.

a one page report is printed. An example of report is given in figure S5. They are all composed of these following parts (from left to right and top to bottom):

- the heading gives the reference of the run (written in blue) and all the parameters of the simulation (written in black).
- up to eight graphic representation of the growth front. Red: G_R, green: G_S, light red: I_R, light green: I_S. They are given in a chronological order, and correspond to a scan of the growth front at regular time interval.
- A three-dimensional representation of the surface of the growth front. This surface is drawn for the same time than the second flat representation of the growth front.
- The evolution of the level of the growth front as a function of simulation time. The red line indicates the lower altitude of the growth front (in number of molecular layers), and the green line the higher altitude of the growth front.
- The enantiomeric excess of each layer of the final crystal.
- The average size of the homochiral structures of each layer.

2 Statistical Fluctuations of Enantiomeric Excess

A system having a finite number n of chiral molecules is subject to statistical fluctuations of its enantiomeric excess *ee*. The evaluation of the value of these fluctuations is a simple combinatorial problem.

2.1 Calculation

Given a system of n molecules, each being of either L or D configuration, the number of possibilities for having $p \in [0, n]$ L and $(n-p)$ D equals:

$$C_n^p = \frac{n!}{(n-p)!p!} \tag{1}$$

The enantiomeric excess of this state is:

$$ee = \frac{(n-p)-p}{n} \tag{2}$$

The average enantiomeric excess for all the possible states of the system is zero (by symmetry). The standard deviation around zero is then:

$$\sqrt{\overline{ee^2}} = \sqrt{\frac{\sum_{p=0}^{n} C_n^p \left(\frac{n-2p}{n}\right)^2}{\sum_{p=0}^{n} C_n^p}} \tag{3}$$

This equation can be calculated using this relation:

$$\sum_{p=0}^{n} C_n^p = 2^n \tag{4}$$

The fluctuations around the racemic state for a system constituted of n molecules is thus:

$$\sqrt{\overline{ee^2}} = \frac{1}{\sqrt{n}} \tag{5}$$

(see Fig. S6 for details)

2.2 Numerical application

The relation 5 implies that for a macroscopic system, fluctuations around the racemic state are low, but may not be negligible. For the simulations that were performed, a common value of n was 65,536 (per layer, that is a square of 256×256 elementary subunits). Enantiomeric excesses in these simulations are therefore significant when somewhat greater than $4 \cdot 10^{-3}$. It should be noted that the observed fluctuations of enantiomeric excesses are well conforming to this value.

For one mole, $n = \mathcal{N} = 6.02 \cdot 10^{23} \Longrightarrow \sqrt{\overline{ee^2}} \approx 1.3 \cdot 10^{-12}$. This value should be compared to the value given for the estimation of the enantiomeric excess induced by PVED[8], i.e. $\varepsilon \in [10^{-17}, 10^{-14}]$, that is a value 100 times to 100,000 times lower. If we want the effect of the parity violation on the enantiomeric excess not to be obstructed by the statistical fluctuations, we would need between $1.7 \cdot 10^4$ and $1.7 \cdot 10^{10}$ moles of compounds, that is in the case of alanine between $1.5 \cdot 10^3$ and $1.5 \cdot 10^9$ kg.

3 Relationship between stochastic and macroscopic rates

3.1 First order reactions

3.1.1 Macroscopic description

For a given reaction $A \rightarrow B$, in a system containing a molecules of A over a total number n of molecules,

[8] Martín Avalos, Reyes Babiano, Pedro Cintas, José L. Jiménez, and Juan C. Palacios. *From parity to chirality: chemical implications revisited* **Tetrahedron: Asymmetry**, 11:2845–2874, 2000.

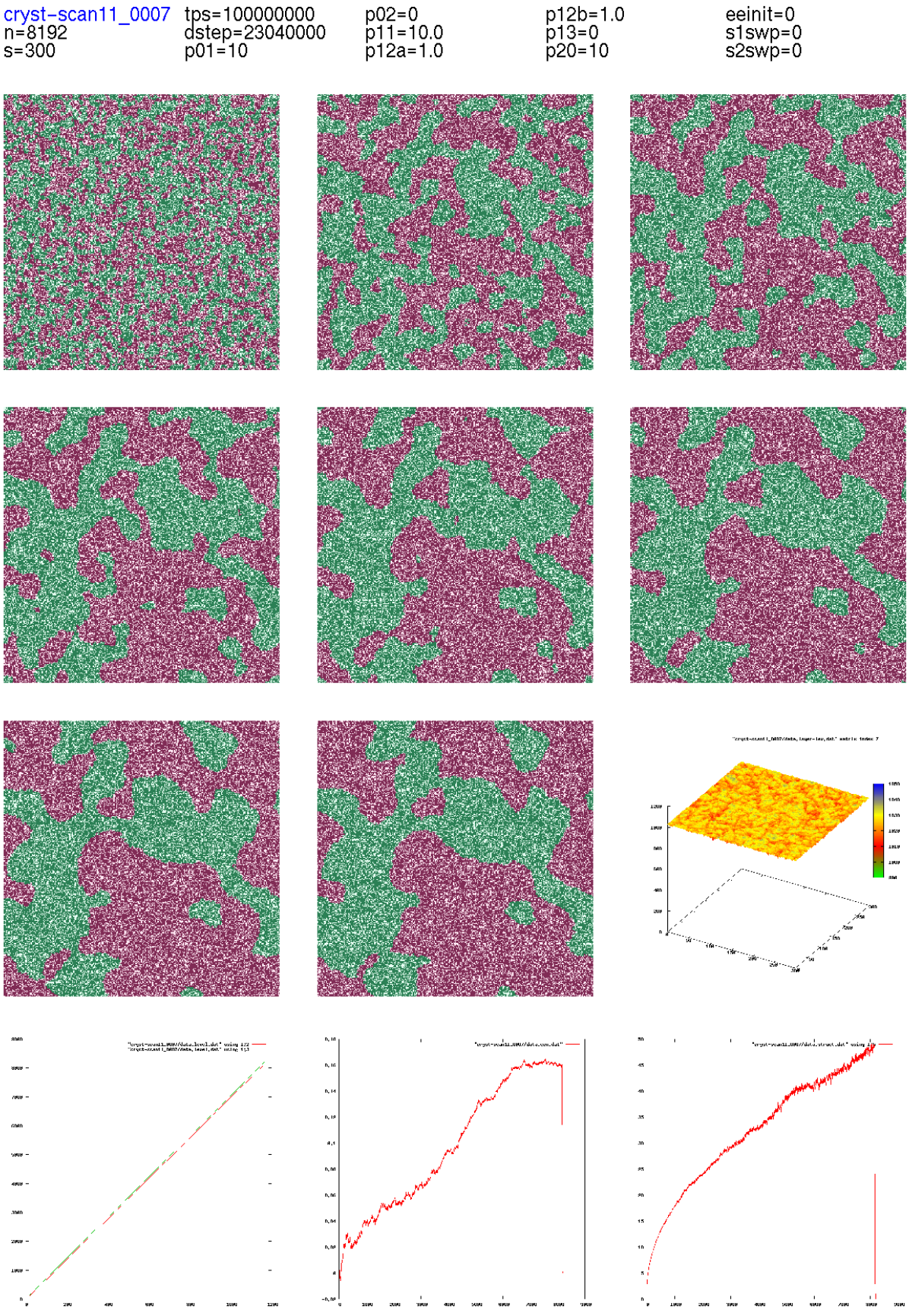

Figure S5: Example of an automatically generated report

$$
\begin{aligned}
\sqrt{\overline{ee^2}} &= \sqrt{\frac{\sum_{p=0}^{n} C_n^p \left(\frac{n-2p}{n}\right)^2}{\sum_{p=0}^{n} C_n^p}} \\
&= \sqrt{2^{-n} \sum_{p=0}^{n} C_n^p \frac{n^2+4p^2-4np}{n^2}} \\
&= \sqrt{2^{-n}\left(2^n + 4\sum_{p=0}^{n} \frac{n!\cdot p^2}{(n-p)!p!\cdot n^2} - 4\sum_{p=0}^{n} \frac{n!\cdot p}{(n-p)!p!\cdot n}\right)} \\
&= \sqrt{1 + 2^{2-n}\left(\sum_{p=1}^{n} \frac{(n-1)!\cdot p}{(n-p)!(p-1)!\cdot n} - \sum_{p=1}^{n} \frac{(n-1)!}{(n-p)!(p-1)!}\right)} \\
&= \sqrt{1 + 2^{2-n}\left(\sum_{p=0}^{n-1} \frac{(n-1)!}{(n-p-1)!p!}\cdot\frac{p+1}{n} - \sum_{p=0}^{n-1} \frac{(n-1)!}{(n-p-1)!p!}\right)} \\
&= \sqrt{1 + 2^{2-n}\left(\sum_{p=0}^{n-1} \frac{p}{n}\cdot C_{n-1}^p + \frac{1}{n}\sum_{p=0}^{n-1} C_{n-1}^p - \sum_{p=0}^{n-1} C_{n-1}^p\right)} \\
&= \sqrt{1 + 2^{2-n}\left(\sum_{p=1}^{n-1} \frac{n-1}{n}\cdot\frac{(n-2)!}{(n-1-p)!(p-1)!} + \frac{2^{n-1}}{n} - 2^{n-1}\right)} \\
&= \sqrt{1 + 2^{2-n}\cdot\frac{n-1}{n}\cdot\sum_{p=0}^{n-2} C_{n-2}^p + \frac{2}{n} - 2} \\
&= \sqrt{\frac{n-1}{n} + \frac{2}{n} - 1} \\
&= \frac{1}{\sqrt{n}}
\end{aligned}
$$

Figure S6: Calculation of the standard deviation of the enantiomeric excess around the racemic state in a system of n molecules.

we have the following parameters:

$$x_a = \frac{a}{n} \quad (6)$$

$$v_a = \frac{dx_a}{dt} \quad (7)$$

$$v_a = k_a x_a \quad (8)$$

where x_a is the molar fraction of A, v_a the rate of variation of A, and k_a the kinetic (macroscopic) rate of A consumption (in s^{-1}).

3.1.2 Stochastic description

In the same system, at a given time, one given molecule of A will react after an average time $\delta t = \frac{1}{p_a}$, p_a being the stochastic rate of A consumption (in s^{-1}). For the subsystem constituted by this unique molecule, we have $\delta a = 1$ during this elementary δt.

For a system constituted by a molecules of A, the total probability of all elementary steps is ap_a. One reaction — corresponding to $\delta a = 1$ — will occur in this system after the elementary time $\delta t = \frac{1}{ap_a}$. We thus obtain:

$$\frac{\delta a}{\delta t} = ap_a \quad (9)$$

$$\Rightarrow \frac{\delta x_a}{\delta t} = x_a p_a \quad (10)$$

At the limit of large values of A, we thus have:

$$p_a = k_a \quad (11)$$

This corresponds to the case $G_x \rightarrow I_x$ in the studied simulations.

3.2 Reaction in a surface site

For a given reaction $A + I \rightarrow B$ (A corresponding to a liquid compound that can react on sites I of a surface) the case is formally identical to the preceding one. As all I sites are surrounded by a A compound (all liquid compounds being identical), this reaction is formally identical to a first order reaction of one $[A + I]$ compound, of molar fraction x_i. Here again, macroscopic and stochastic rates are equal. This corresponds to the cases $L + I_x \rightarrow G_x + C_x$.

3.3 Second order reaction in a surface site

3.3.1 Macroscopic description

For a given reaction $A + I + C \rightarrow B$ (A being a liquid phase surrounding all sites) corresponding to a reaction on a site I, catalyzed by a neighboring site C, we have the following parameters:

$$S = \frac{\mathcal{N}s}{n} \quad (12)$$

$$x_i = \frac{i}{n} \quad (13)$$

$$x_c = \frac{c}{n} \quad (14)$$

$$\sigma_i = \frac{x_i}{S} \quad (15)$$

$$\sigma_c = \frac{x_c}{S} \quad (16)$$

$$v_i = \frac{d\sigma_i}{dt} \quad (17)$$

$$= k_{ic}\sigma_i\sigma_c \quad (18)$$

where S is the molar surface of the crystal (i.e. the surface occupied by one mole of sites), $\mathcal{N}$ the Avogadro number, s the total surface of the crystal, n the number of sites on the surface s, σ_i and σ_c the surfacic concentration of respectively I and C, i and c the number of sites I and C on the surface s, x_i and x_c the fraction of sites occupied by I and C, v_i the rate of variation of I, and k_{ic} the kinetic (macroscopic) rate of I consumption (in $mol^{-1}.m^2.s^{-1}$). This kinetic rates expression makes implicitly the assumption that there is an homogeneous distribution of sites on the crystal surface.

3.3.2 Stochastic description

For a given subsystem constituted by one site on the surface of the total system, the probability of reaction is:

$$p = \begin{cases} 0 & \text{site not occupied by I} \\ 0 & \text{site occupied by I, no neighboring C} \\ np_{ic} & \text{site occupied by I, with neighboring C} \end{cases} \quad (19)$$

The probability that one given neighboring site of the subsystem is occupied by C is $\frac{c}{n}$. Thus, *assuming that the sites are uniformly distributed on the surface*, the total probability of the subsystem to react is:

$$p = 4\frac{c}{n}p_{ic} \quad (20)$$

One should notice that this expression is actually an approximation. Actually, if a first neighboring site is occupied by C, the probability that a second neighboring site is occupied by C will be $\frac{c-1}{n}$. The total probability should thus become:

$$p = \frac{4c-6}{n}p_{ic} \quad (21)$$

However, the correction term is proportional to n^{-1}, and will become negligible for a macroscopic system. Thus, only the equation 20 will be considered.

For the whole system, the total probability of all elementary steps is then:

$$p = 4\frac{ic}{n}p_{ic} \quad (22)$$

$$= \frac{\delta i}{\delta t} \quad (23)$$

$$\Rightarrow \frac{\delta \sigma_i}{\delta t} = \frac{1}{ns}\frac{\delta i}{\delta t} \quad (24)$$

$$= 4S\sigma_i\sigma_c p_{ic} \quad (25)$$

At the limit of large dimension of an homogeneous system, we thus have:

$$k_{ic} = 4Sp_{ic} \quad (26)$$

k_{ic} is in $mol^{-1}.m^2.s^{-1}$, S in $mol^{-1}.m^2$ and p_{ic} in s^{-1}. This corresponds to the case $L + I_x + G_y \rightarrow G_z + C_x + G_y$ in the studied simulations.

4 Program Listing

4.1 ODE file of the homogeneous model

This is the `.ode` file used for the numeric integration of the homogeneous model of frontal crystallization by the `XppAut` software.

```
ir'=r14-r12-r10-r9-r6-r4-r2
is'=r13-r11-r8-r7-r5-r3-r1
gr'=r2+r3+r6+r8+r10+r11-r14
gs'=r1+r4+r5+r7+r9+r12-r13

r1=k01*is
r2=k01*ir
r3=k02*is
r4=k02*ir
r5=k11*is*gs
r6=k11*ir*gr
r7=k12*is*gr
r8=k12*is*gr
r9=k12*ir*gs
r10=k12*ir*gs
r11=k13*is*gs
r12=k13*ir*gr
r13=k20*gs
r14=k20*gr

k01=0.001
k12=0.002
k20=0.005
k11=a*0.001
k02=b*1e-6
k13=b*1e-6

aux ee1=(gs-gr)/(gr+gs)
aux ee2=(is-ir)/(ir+is)

par a=5, b=1

init gs=0.26, gr=0.24, is=0.25, ir=0.25
@ meth=qualrk, total=500000, dt=200, xhi=500000, nplot=2, yp=ee1, yp2=ee2, ylo=0, yhi=1
done
```

4.2 Main Program

4.2.1 include.h

There are general definitions and includes.

#ifndef INCLUDE_H

```
#define INCLUDE_H

#include <stdlib.h>       //Use of random function

#define ESP_L 0     //don't change values (used in calc)
#define ESP_GS 1
#define ESP_GR 2
#define ESP_IS 3
#define ESP_IR 4
#define ESP_CS 5
#define ESP_CR 6

#define REAC_20 0   // same def of reac as in jpcb-05
#define REAC_01 1
#define REAC_02 2
#define REAC_11 3
#define REAC_12a 4
#define REAC_12b 5
#define REAC_13 6

#define MAX_REAC_ONE_POINT 10
#define TMAX 10   // dt chosen in the case of no reaction ar further possible

#define FILEMAX 50
#define NB_OUT_FILE 7

struct Reac {
   int s;
   int n;
   int s2;
   char type;
   double proba;
   char next;
};

#endif
```

4.2.2 cryst.cpp

This is the main program, that initiates calculations, read the parameters file and generates the output files.

```
#include "medium.h"
using namespace std;

int main (int argc, char * argv[]){
   if (argc==3)
      {
         char filename[FILEMAX];
         char *suff[NB_OUT_FILE];
         suff[0]=".layer-lev.dat";
         suff[1]=".layer-spec.dat";
         suff[2]=".slice.dat";
         suff[3]=".een.dat";
         suff[4]=".level.dat";
         suff[5]=".struct.dat";
         suff[6]=".struct2.dat";
         ofstream **output;
         output=(ofstream**)malloc(NB_OUT_FILE*sizeof(ofstream*));
         int n,s,dstep;
         double tps,t;
         double k[10];
```

```
    Medium *medium;
    //seed from urandom
    char data[4];
    long seed;
    ifstream random ("/dev/urandom", ios::in);
    random.read(data, 4*sizeof(char));
    seed=data[0]+128+(data[1]+128)*256+(data[2]+128)*256*256+(data[3]+128)*256*256*256;
    srand48(seed);
    random.close();

    ifstream param(argv[1], ios::in);
    for (int i=0; i<NB_OUT_FILE; i++)
      {
        strcpy(filename,argv[2]);
        strcat(filename,suff[i]);
        *(output+i)=new ofstream(filename, ios::out);
      }
    //read parameters
    param >> n;
    cout << "n=" << n << "\n";
    param >> s;
    cout << "s=" << s << "\n";
    param >> tps;
    cout << "tps=" << tps << "\n";
    param >> dstep;
    cout << "dt=" << dstep << "\n";
    param >> k[REAC_01];
    cout << "k01=" << k[REAC_01] << "\n";
    param >> k[REAC_02];
    cout << "k02=" << k[REAC_02] << "\n";
    param >> k[REAC_11];
    cout << "k11=" << k[REAC_11] << "\n";
    param >> k[REAC_12a];
    cout << "k12a=" << k[REAC_12a] << "\n";
    param >> k[REAC_12b];
    cout << "k12b=" << k[REAC_12b] << "\n";
    param >> k[REAC_13];
    cout << "k13=" << k[REAC_13] << "\n";
    param >> k[REAC_20];
    cout << "k20=" << k[REAC_20] << "\n";
    param >> k[7];
    cout << "ee_init=" << k[7] << "\n";
    param >> k[8];
    cout << "s1swp=" << k[8] << "\n";
    param >> k[9];
    cout << "s2swp=" << k[9] << "\n";
    // Initialize simulator
    medium=new Medium(n,s,k);
    // Process
    for (t=0;t<tps && medium->run_reac;t+=medium->process(dstep,output));
    cout << "Run OK\n";
    medium->ee_out(*(output+3));
    medium->slice_out(*(output+2));
    medium->struct_out(*(output+5));
    medium->struct2_out(*(output+6));
    for (int i=0; i<NB_OUT_FILE; i++) (*(output+i))->close();
    free(output);
    delete medium;
    cout << "End of Program :-)\n";
    return 0;
}
```

```
  else
    {
      cout << "usage: 'cryst param-file output-file'\n";
      return 2;
    }
}
```

4.2.3 Makefile

Makefile used for the generation of the executable file program.

```
CFLAGS = -s -O3 -funroll-loops -march=pentium4 -Wall
#CFLAGS = -g -pg -Wall

cryst : cryst.o medium.o layer.o sublayer.o event.o
        g++ $(CFLAGS) medium.o event.o layer.o sublayer.o cryst.o -o cryst
event.o : event.cpp medium.h event.h layer.h
        g++ $(CFLAGS) -c $*.cpp
layer.o : layer.cpp medium.h event.h layer.h sublayer.h
        g++ $(CFLAGS) -c $*.cpp
sublayer.o : sublayer.cpp medium.h event.h layer.h sublayer.h
        g++ $(CFLAGS) -c $*.cpp
medium.o : medium.cpp medium.h event.h layer.h
        g++ $(CFLAGS) -c $*.cpp
cryst.o : cryst.cpp medium.h event.h layer.h
        g++ $(CFLAGS) -c $*.cpp
clean:
        rm -v *.o *~
```

4.2.4 batch.py

This is the batch python program, used for calling the calculation program for several set of parameters, generating and printing a report page for each performed calculation, and compressing all the output files in a single `.tar.bz2` archive file.

```
import os,sys
import string
import Image
import ImageDraw
import ImageFont
import Gnuplot

col=[(255,255,255),(43,128,85),(128,43,85),(230,255,242),(255,230,242),(168,255,211),(255,168,211),(0,0,0)]
subproj_num=0

'''Simulation parameters'''
n=8192
s=256
tps=100000000
dstep=10000
p01=0.1
p02=0.01
p11=10.0
p12a=4.5
p12b=4.5
p13=0.01
p20=10
ee_init=0
s1swp=0
s2swp=0
image_aff=["1","13","25","38","50","63"]
```

```
layer_lev="7"
project_name="cryst-scan28"

for p12a in (3.6,):
        p12b=p12a
        for ee_init in (0,):
            for rep in range(2):
                dstep=s*s*n/32

                '''Create param file'''
                subproj_num=subproj_num+1
                subproj_name="00000000"+str(subproj_num)
                subproj_name=project_name+"_"+subproj_name[-4:]
                os.mkdir(subproj_name)
                print("\n\n-----------------------------------------\n")
                os.system("date")
                print("Subproject "+subproj_name+" started.\n")
                param_file=open(subproj_name+"/param","w")
                param_file.write(str(n)+"\n")
                param_file.write(str(s)+"\n")
                param_file.write(str(tps)+"\n")
                param_file.write(str(dstep)+"\n")
                param_file.write(str(p01)+"\n")
                param_file.write(str(p02)+"\n")
                param_file.write(str(p11)+"\n")
                param_file.write(str(p12a)+"\n")
                param_file.write(str(p12b)+"\n")
                param_file.write(str(p13)+"\n")
                param_file.write(str(p20)+"\n")
                param_file.write(str(ee_init)+"\n")
                param_file.write(str(s1swp)+"\n")
                param_file.write(str(s2swp)+"\n")
                param_file.write('\nFile automatically generated.\nProject "'+subproj_name+'".\n')
                param_file.close()

                '''Calculation'''
                os.system("unbuffer ./cryst "+subproj_name+"/param "+subproj_name+"/data ")

                '''Process images'''
                infile=subproj_name+"/data.layer-spec.dat"
                fin=open(infile,"r")
                image=Image.new("RGB",(s,s))
                k=0
                rdl=fin.readline()
                try:
                    while rdl!='':
                        k=k+1
                        kstr="0000000000"+str(k)
                        try:
                            for i in range(s):
                                ligne=fin.readline()
                                liste=string.split(ligne)
                                for j in range(s):
                                    color=int(liste[j])
                                    if (color>6 or color<0):
                                        color=7
                                    image.putpixel((i,s-j-1),col[color])
                        except:
                            print('Woops!!!!')
                            print('i:')
                            print(i)
```

```
                print(' j:')
                print(j)
                print(' k:')
                print(k)
                print('col')
                print(int(liste[j]))
            image.save(infile+"."+kstr[-6:]+".png","PNG")
            print("("+kstr[-6:]+") "+rdl[:-1]+" : OK!\n")
            rdl=fin.readline()
except:
    print("Program incorrectly stopped, couldn't finished last image creation !!!\n")
del(image)
fin.close()

os.system("/usr/bin/convert "+subproj_name+"/*.png "+subproj_name+"/anim.gif")

infile=subproj_name+"/data.slice.dat"
fin=open(infile,"r")
image=Image.new("RGB",(s,n))
try:
    try:
        for i in range(s):
            ligne=fin.readline()
            liste=string.split(ligne)
            for j in range(n):
                color=int(liste[j])
                if (color>6 or color<0):
                    color=7
                image.putpixel((i,n-j-1),col[color])
    except:
        print('Woops!!!!')
        print('i:')
        print(i)
        print(' j:')
        print(j)
        print(' k:')
        print(k)
        print('col')
        print(int(liste[j]))
    image.save(subproj_name+"/slice.png","PNG")
    print("Slice : OK!\n")
    rdl=fin.readline()
except:
    print("Program incorrectly stopped, couldn't finished last image creation !!!\n")
del(image)
fin.close()

'''Create figs'''
plot=Gnuplot.Gnuplot()
plot('set terminal png size 700,700')
'''levels'''
plot('set output "'+subproj_name+'/levels.png"')
plot('plot "'+subproj_name+'/data.level.dat" using 1:2 with lines, "'+subproj_name+\
      '/data.level.dat" using 1:3 with lines')
plot('unset output')
'''een'''
plot('set output "'+subproj_name+'/een.png"')
plot('plot "'+subproj_name+'/data.een.dat" with lines')
plot('unset output')
'''struct'''
plot('set output "'+subproj_name+'/struct.png"')
```

```
plot('plot "'+subproj_name+'/data.struct.dat" using 1:5 with lines')
plot('unset output')
'''surface'''
plot('set output "'+subproj_name+'/surface.png"')
plot('set pm3d')
plot('set ticslevel 0')
plot('set palette defined ( 0 "green" , 1 "red", 2 "yellow" , 3 "blue")')
plot('splot [0:'+str(s)+'] [0:'+str(s)+'] [0:] "'+subproj_name+\
      '/data.layer-lev.dat" matrix index '+layer_lev+'with pm3d')
plot('unset output')
'''struct2'''
plot('set output "'+subproj_name+'/struct2.png"')
plot('set logscale')
plot('plot "'+subproj_name+'/data.struct2.dat" index '+str(n-1)+\
      ' using  ((sgn($1)+1)/2*$1):(abs($2)) with lines smooth unique, "'\
      +subproj_name+'/data.struct2.dat" index '+str(n-1)+\
      ' using  ((sgn($1)-1)/2*$1):(abs($2)) with lines smooth unique, "'\
      +subproj_name+'/data.struct2.dat" index '+str(n)+\
      ' using  ((sgn($1)+1)/2*$1):(abs($2)) with lines smooth unique, "'\
      +subproj_name+'/data.struct2.dat" index '+str(n)+\
      ' using  ((sgn($1)-1)/2*$1):(abs($2)) with lines smooth unique')
plot('unset output')
plot('exit')
del(plot)

'''Create report'''
report=Image.new("RGB",(2480,3508),(255,255,255))
'''n=0'''
kstr="0000000000"+image_aff[0]
try:
     im=Image.open(subproj_name+'/data.layer-spec.dat.'+kstr[-6:]+'.png')
     scaled=im.resize((700,700))
     report.paste(scaled,(95,328,795,1028))
except IOError:
     print(kstr[-6:]+" not here\n")
'''n=1'''
kstr="0000000000"+image_aff[1]
try:
     im=Image.open(subproj_name+'/data.layer-spec.dat.'+kstr[-6:]+'.png')
     scaled=im.resize((700,700))
     report.paste(scaled,(890,328,1590,1028))
except IOError:
     print(kstr[-6:]+" not here\n")
'''n=2'''
kstr="0000000000"+image_aff[2]
try:
     im=Image.open(subproj_name+'/data.layer-spec.dat.'+kstr[-6:]+'.png')
     scaled=im.resize((700,700))
     report.paste(scaled,(1685,328,2385,1028))
except IOError:
     print(kstr[-6:]+" not here\n")
'''n=3'''
kstr="0000000000"+image_aff[3]
try:
     im=Image.open(subproj_name+'/data.layer-spec.dat.'+kstr[-6:]+'.png')
     scaled=im.resize((700,700))
     report.paste(scaled,(95,1123,795,1823))
except IOError:
     print(kstr[-6:]+" not here\n")
'''n=4'''
kstr="0000000000"+image_aff[4]
```

```
try:
    im=Image.open(subproj_name+'/data.layer-spec.dat.'+kstr[-6:]+'.png')
    scaled=im.resize((700,700))
    report.paste(scaled,(890,1123,1590,1823))
except IOError:
    print(kstr[-6:]+" not here\n")
'''n=5'''
kstr="0000000000"+image_aff[5]
try:
    im=Image.open(subproj_name+'/data.layer-spec.dat.'+kstr[-6:]+'.png')
    scaled=im.resize((700,700))
    report.paste(scaled,(1685,1123,2385,1823))
except IOError:
    print(kstr[-6:]+" not here\n")
'''slice'''
try:
    im=Image.open(subproj_name+'/slice.png')
    scaled=im.resize((700,700))
    report.paste(scaled,(95,1918,795,2618))
except IOError:
    print("Slice not here\n")
'''struct2'''
try:
    im=Image.open(subproj_name+'/struct2.png')
    report.paste(im,(890,1918,1590,2618))
except IOError:
    print("Struct2 not here\n")
'''report.save("/tmp/1.png","PNG")'''
'''levels'''
try:
    im=Image.open(subproj_name+'/levels.png')
    report.paste(im,(95,2713,795,3413))
except IOError:
    print("levels not here\n")
'''een'''
try:
    im=Image.open(subproj_name+'/een.png')
    report.paste(im,(890,2713,1590,3413))
except IOError:
    print("een not here\n")
'''struct'''
try:
    im=Image.open(subproj_name+'/struct.png')
    report.paste(im,(1685,2713,2385,3413))
except IOError:
    print("struct not here\n")
'''surface'''
try:
    im=Image.open(subproj_name+'/surface.png')
    report.paste(im,(1685,1918,2385,2618))
except IOError:
    print("surface not here\n")
del(im)
del(scaled)
'''finish report'''
draw=ImageDraw.Draw(report)
fnt=ImageFont.load("helv.pil")
draw.text((95,95),subproj_name,fill=(0,0,255),font=fnt)
draw.text((95,141),"n="+str(n),fill=(0,0,0),font=fnt)
draw.text((95,187),"s="+str(s),fill=(0,0,0),font=fnt)
draw.text((553,95),"tps="+str(tps),fill=(0,0,0),font=fnt)
```

```
                draw.text((553,141),"dstep="+str(dstep),fill=(0,0,0),font=fnt)
                draw.text((553,187),"p01="+str(p01),fill=(0,0,0),font=fnt)
                draw.text((1011,95),"p02="+str(p02),fill=(0,0,0),font=fnt)
                draw.text((1011,141),"p11="+str(p11),fill=(0,0,0),font=fnt)
                draw.text((1011,187),"p12a="+str(p12a),fill=(0,0,0),font=fnt)
                draw.text((1469,95),"p12b="+str(p12b),fill=(0,0,0),font=fnt)
                draw.text((1469,141),"p13="+str(p13),fill=(0,0,0),font=fnt)
                draw.text((1469,187),"p20="+str(p20),fill=(0,0,0),font=fnt)
                draw.text((1927,95),"eeinit="+str(ee_init),fill=(0,0,0),font=fnt)
                draw.text((1927,141),"s1swp="+str(s1swp),fill=(0,0,0),font=fnt)
                draw.text((1927,187),"s2swp="+str(s2swp),fill=(0,0,0),font=fnt)
                report.save(subproj_name+'/report.png',"PNG")
                del(report)
                os.system("lpr -o dpi=600 -o media=A4 -o scaling=100 "+subproj_name+"/report.png")
                os.system("tar -cvj "+subproj_name+"/ -f "+subproj_name+".tar.bz2")
                os.system("rm -r "+subproj_name)
print("Batch ran smoothly :-)")
```

4.3 Medium Class

This class contain the 3-D matrix describing the growing crystal, and its link (via layer and sublayer classes) with possible reactions (described in event class).

4.3.1 medium.h

```
#ifndef MEDIUM_H
#define MEDIUM_H

#include "include.h"
#include "layer.h"
#include <iostream>
#include <fstream>
using namespace std;

class Layer;
class Reac;
class Linked_node;

class Medium {
  // Attributes
private:
  double time;
  int nmax;
  int smax;
  char* mat;
  int* lev;
  double probas[7];
  Layer *layer;
  bool *mask;
  Linked_node **node;
public:
  bool run_reac;
  // Operations
public:
  Medium(int , int, double*);
  ~Medium();
  void make_reac ( Reac * reac );
  double process ( int, ofstream **);
  char what(int,int,int);
  Linked_node* what_node(int,int,int);
  double proba(char);
```

```
    void ee_out(ofstream *);
    void struct_out(ofstream *);
    void slice_out(ofstream *);
    void struct2_out(ofstream *);
private:
    int size_struct(int,int,int);
    void size_struct_3d(int,int,int);
    int category(long);
    long num_cat(int);
};

#endif
```

4.3.2 medium.cpp

```
#include "medium.h"

class Linked_node{
    // Attributes
public:
    int value;
    Linked_node *next;
    // Operations
public:
    Linked_node(int val){
        value=val;
        next=NULL;
    }
    ~Linked_node(){
        if (next != NULL)
            {
                delete next;
                next=NULL;
            }
    }
    Linked_node *add(){
        if (next==NULL)
            {
                if (value>0)
                    value++;
                else
                    value--;
                return this;
            }
        else return next->add();
    }
    void merge(Linked_node *other){
        if (next==NULL)
            {
                while (other->next != NULL) other=other->next;
                if (other!=this)
                    {
                        next=other->next=new Linked_node(value+other->value);
                        value=0;
                        other->value=0;
                    }
            }
        else next->merge(other);
    }
};
```

```
Medium::Medium(int n, int s, double * rates){
  int i,j,k;
  double tmp2,tmp3;
  time=0;
  nmax=n;
  smax=s;
  run_reac=true;
  for (i=0; i<7; i++) probas[i]=*(rates+i);
  mat=new char[n*s*s];
  mask=new bool[s*s];
  double tmp=(*(rates+7)+1)/2;
  tmp3=tmp2=tmp;
  for(j=0; j<smax; j++)
    {
      if (*(rates+8)==1 && j>=smax/2) tmp3=tmp2=(1-tmp);
      for(k=0; k<smax; k++)
        {
          if (*(rates+9)==1 && k>=smax/2) tmp3=1-tmp2;
          *(mat+j*smax+k)=(drand48()<tmp3 ? ESP_IR : ESP_IS);
        }
      tmp3=tmp2;
    }
  for(i=1; i<nmax; i++)
    for(j=0; j<smax; j++)
      for(k=0; k<smax; k++)
        *(mat+i*smax*smax+j*smax+k)=ESP_L;
  lev=new int[smax*smax];
  for(j=0; j<smax; j++)
    for(k=0; k<smax; k++)
      *(lev+j*smax+k)=1;
  layer=new Layer(s, this);
  cout << "Medium Created!\n";
}

Medium::~Medium(){
  delete[] lev;
  delete[] mask;
  delete   layer;
  delete[] mat;
  cout << "Medium Destroyed!\n";
}

void Medium::make_reac ( Reac* reac ){
  int n=reac->n;
  int s=reac->s;
  int s2=reac->s2;
  *(mat+n*smax*smax+s*smax+s2)=reac->next;
  if (reac->type == REAC_20)
    *(lev+s*smax+s2)+=1;
  else
    *(mat+(n-1)*smax*smax+s*smax+s2)+=2;
  if (*(lev+s*smax+s2)==nmax)
    run_reac=false;
  else
    {
      layer->update(*(lev+s*smax+s2),s,s2);
      if (s>0)
        layer->update(*(lev+(s-1)*smax+s2),s-1,s2);
      else
        layer->update(*(lev+(smax-1)*smax+s2),smax-1,s2);
      if (s<(smax-1))
```

```
            layer->update(*(lev+(s+1)*smax+s2),s+1,s2);
         else
            layer->update(*(lev+s2),0,s2);
         if (s2>0)
            layer->update(*(lev+s*smax+s2-1),s,s2-1);
         else
            layer->update(*(lev+s*smax+smax-1),s,smax-1);
         if (s2<(smax-1))
            layer->update(*(lev+s*smax+s2+1),s,s2+1);
         else
            layer->update(*(lev+s*smax),s,0);
      }
}

double Medium::process ( int max_step, ofstream** output ){
   double t=0;
   int levelmin,levelmax,tmp;
   for (int i=0;i<max_step && run_reac ;i++)
      t+=layer->process_reac();
   time+=t;
   cout << "t=" << time << "\n";
   **output << "\n";
   **(output+1) << "\n";
   levelmin=levelmax=*lev;
   for (int j=0; j<smax; j++)
      {
         for (int k=0; k<smax; k++)
            {
               int level=*(lev+j*smax+k);
               if (levelmin>level) levelmin=level;
               if (levelmax<level) levelmax=level;
               **output << level << " ";
               tmp=(int)(*(mat+level*smax*smax+j*smax+k));
               if (tmp==0)    tmp=(int)(*(mat+(level-1)*smax*smax+j*smax+k));
               **(output+1) <<    tmp << " ";
            }
         **output << "\n";
         **(output+1) << "\n";
      }
   **(output+4) << time << " " << levelmin << " "
                     << levelmax << " " << "\n";
   return t;
}

void Medium::slice_out (ofstream *output){
   for (int j=0; j<smax; j++)
      {
         for (int i=0; i<nmax; i++)
            *output << (int)(*(mat+i*smax*smax+j*smax+smax/2)) << " ";
         *output << "\n";
      }
}

char Medium::what (int n, int s, int s2){
   if (s<0) s+=smax;
   if (s2<0) s2+=smax;
   if (n<0) n+=nmax;
   if (s>=smax) s-=smax;
   if (s2>=smax) s2-=smax;
   if (n>=nmax) n-=nmax;
   return *(mat+n*smax*smax+s*smax+s2);
```

```
}

Linked_node *Medium::what_node (int n, int s, int s2){
  if (s<0) s+=smax;
  if (s2<0) s2+=smax;
  if (n<0) n+=nmax;
  if (s>=smax) s-=smax;
  if (s2>=smax) s2-=smax;
  if (n>=nmax) n-=nmax;
  Linked_node* tmp=*(node+n*smax*smax+s*smax+s2);
  if (tmp!=NULL)
    while (tmp->next!=NULL)
      tmp=tmp->next;
  return tmp;
}

double Medium::proba (char type){  return probas[(int)type]; }

void Medium::ee_out (ofstream * output){
  for (int i=0; i<nmax; i++)
    {
      int tmp=0;
      for (int j=0; j<smax; j++)
        for (int k=0; k<smax; k++)
          {
            char tmp2=*(mat+i*smax*smax+j*smax+k);
            if (tmp2 != 0)
              tmp+=((tmp2%2 == 0) ? 1 : -1);
          }
      *output << i << " " << (double)tmp/(double)(smax*smax) << "\n";
    }
}

void Medium::struct_out (ofstream * output){
  for (int i=0; i<nmax; i++)
    {
      int total=0;
      int sum=0;
      int contact=0;
      for (int j=0; j<smax; j++)
        for (int k=0; k<smax; k++)
          {
            char current=what(i,j,k);
            if (current!=0)
              {
                current %= 2;
                sum+=current;
                total++;
                char rightside=what(i,j+1,k);
                if (rightside != 0)
                  {
                    rightside %= 2;
                    contact+=(current != rightside);
                  }
                else
                  contact++;
                char downside=what(i,j,k+1);
                if (downside != 0)
                  {
                    downside %= 2;
                    contact+=(current != downside);
```

```
                        }
                    else
                        contact++;
                }
            }
        sum*=2;
        if (sum>total) sum=2*total-sum;
        double radius=(double)sum/(double)contact;
        *output << i << " " << total << " " << sum/2 << " " << contact
                << " "<< 2*radius << " " << 3.14*radius*radius << "\n";
    }
}
int Medium::size_struct(int n, int s, int s2){
  if (s<0) s+=smax;
  if (s2<0) s2+=smax;
  if (n<0) n+=nmax;
  if (s>=smax) s-=smax;
  if (s2>=smax) s2-=smax;
  if (n>=nmax) n-=nmax;
  if  (*(mask+s*smax+s2))
    return 0;
  else
    {
      *(mask+s*smax+s2)=true;
      char center=*(mat+n*smax*smax+s*smax+s2);
      if (center < ESP_CS)
        return 0;
      else
        {
          int res=2*center-11;  // i.e. -1 for ESP_CS and +1 for ESP_CR
          if (center==what(n,s,s2+1)) res+=size_struct(n,s,s2+1);
          if (center==what(n,s,s2-1)) res+=size_struct(n,s,s2-1);
          if (center==what(n,s+1,s2)) res+=size_struct(n,s+1,s2);
          if (center==what(n,s-1,s2)) res+=size_struct(n,s-1,s2);
          return res;
        }
    }
}

void Medium::size_struct_3d(int n, int s, int s2){
  Linked_node *tmp2=NULL;
  char center=*(mat+n*smax*smax+s*smax+s2);
  if (center < ESP_CS)
    tmp2=new Linked_node(0);
  else
    {
      Linked_node *tmp=what_node(n,s,s2-1);
      if (tmp != NULL && center==what(n,s,s2-1))
        tmp2=tmp->add();
      tmp=what_node(n,s,s2+1);
      if (tmp !=NULL && center==what(n,s,s2+1))
        if (tmp2 == NULL)
          tmp2=tmp->add();
        else
          tmp2->merge(tmp);
      tmp=what_node(n,s-1,s2);
      if (tmp !=NULL && center==what(n,s-1,s2))
        if (tmp2 == NULL)
          tmp2=tmp->add();
        else
          tmp2->merge(tmp);
```

```
            tmp=what_node(n,s+1,s2);
            if (tmp !=NULL && center==what(n,s+1,s2))
               if (tmp2 == NULL)
                  tmp2=tmp->add();
               else
                  tmp2->merge(tmp);
            tmp=what_node(n-1,s,s2);
            if (tmp !=NULL && center==what(n-1,s,s2))
               if (tmp2 == NULL)
                  tmp2=tmp->add();
               else
                  tmp2->merge(tmp);
            if (tmp2 == NULL)
               tmp2=new Linked_node(2*center-11);//i.e. -1 for ESP_CS and 1 for ESP_CR
         }
      *(node+n*smax*smax+s*smax+s2)=tmp2;
}

int Medium::category(long n){
   int s;
   if (n<0)
      {
         n=-n;
         s=-1;
      }
   else
      s=1;
   int z=(int)floor(log10((double)n));
   int y=(int)floor(n/pow(10,(double)z));
   return (10*z+y)*s;
}

long Medium::num_cat(int cat){
   int s;
   if (cat<0)
      {
         cat=-cat;
         s=-1;
      }
   else
      s=1;
   int l=cat/10;
   int m=cat%10;
   return (long)(s*(m+0.5)*pow(10.,l));
}

void Medium::struct2_out(ofstream *output){
   int i,j,k;
   int cat1=category(smax*smax);
   int cat2=category(smax*smax*nmax);
   int tmpp[cat1+1];
   int tmpm[cat1+1];
   int tmp2p[cat1+1];
   int tmp2m[cat1+1];
   int tmp3p[cat2+1];
   int tmp3m[cat2+1];
   int size;
   for (j=0; j<cat1+1; j++)
      {
         tmpp[j]=0;
         tmp2p[j]=0;
```

```
        tmpm[j]=0;
        tmp2m[j]=0;
    }
  for (j=0; j<cat2+1; j++)
    {
        tmp3p[j]=0;
        tmp3m[j]=0;
    }
  for (i=0; i<nmax; i++)
    {
        for (j=0; j<smax*smax; j++) *(mask+j)=false;
        for (j=0; j<smax; j++)
            for (k=0; k<smax; k++)
                {
                    size=category(size_struct(i,j,k));
                    if (size<0)
                        tmpm[-size]++;
                    else
                        tmpp[size]++;
                }
        for (j=1; j<cat1+1; j++)
            {
                if (tmpp[j]!=0)
                    {
                        *output << num_cat(j) << " " << tmpp[j]/pow(10.,j/10) << "\n";
                        tmp2p[j]+=tmpp[j];
                        tmpp[j]=0;
                    }
                if (tmpm[j]!=0)
                    {
                        *output << -num_cat(j) << " " << tmpm[j]/pow(10.,j/10) << "\n";
                        tmp2m[j]+=tmpm[j];
                        tmpm[j]=0;
                    }
            }
        *output << "\n\n";
    }
  for (j=1; j<cat1+1; j++)
    {
        if (tmp2p[j]!=0)
            *output << num_cat(j) << " " << tmp2p[j]/pow(10.,j/10) << "\n";
        if (tmp2m[j]!=0)
            *output << -num_cat(j) << " " << tmp2m[j]/pow(10.,j/10) << "\n";
    }
*output << "\n\n";
node=new (Linked_node*)[nmax*smax*smax];
for (i=0; i<nmax*smax*smax; i++) *(node+i)=NULL;
for (i=0; i<nmax; i++)
    for (j=0; j<smax; j++)
        for (k=0; k<smax; k++)
            size_struct_3d(i,j,k);
for (i=0; i<nmax*smax*smax; i++)
    {
        size=category((*(node+i))->value);
        (*(node+i))->value=0;
        if (size<0)
            tmp3m[-size]++;
        else
            tmp3p[size]++;
    }
for (j=1; j<cat2+1; j++)
```

```
    {
      if (tmp3p[j]!=0)
        *output << num_cat(j) << " " << tmp3p[j]/pow(10.,j/10) << "\n";
      if (tmp3m[j]!=0)
        *output << -num_cat(j) << " " << tmp3m[j]/pow(10.,j/10) << "\n";
    }
}
```

4.4 Layer and Sublayer Class

These classes are the link between medium class and event class. An initially more general layer class have been splitted into these two layer and sublayer classes for calculation time optimization purpose. They permit to choose from the medium class which possible reaction (one instance of event class) will occur.

4.4.1 layer.h

```
#ifndef LAYER_H
#define LAYER_H

#include "sublayer.h"
#include "medium.h"
#include "include.h"

class Medium;

class Sublayer;

class Layer {
  // Attributes
 private:
  Medium *medium;
  Sublayer **sublayers;
  int smax;
  double proba_max;
  // Operations
 public:
  Layer (int, Medium*);
  ~Layer();
  double process_reac ( );
  void update ( int n, int s , int s2);
};

#endif
```

4.4.2 layer.cpp

```
#include "layer.h"

Layer::Layer(int s, Medium * parent){
  int i,j;
  smax=s;
  medium=parent;
  proba_max=0;
  sublayers=new Sublayer*[smax];
  for (i=0; i<smax; i++)
    {
      (*(sublayers+i))=new Sublayer(smax,i,medium);
      for (j=0; j<smax;j++) update(1,i,j);
    }
}
```

```
Layer::~Layer(){
  for (int i=0; i<smax; i++) delete *(sublayers+i);
  delete[] sublayers;
}

double Layer::process_reac ( ){
  double rand;
  Reac* result;
  do rand=drand48()*proba_max;
  while (rand==proba_max);  //uniform random number in [0,proba_max[
  double tmp2=0, tmp3=0;
  int i=0;
  double dt=(proba_max==0 ? TMAX : 1/proba_max);
  do
    {
      tmp3=tmp2;
      tmp2+=(*(sublayers+i))->proba_tot;
      i++;
    }
  while(rand>=tmp2 && i<smax);
  if (rand==tmp2)
    {
      cout << "bounding problem in layer ! rand="<< rand
           << " proba_max=" << proba_max << "\n";
      return 0;
    }
  result=(*(sublayers+i-1))->choose_reac(rand-tmp3);
  if (result)
    medium->make_reac(result);
  else
    {
      cout <<  "layer " << i << " !\n";
      return 0;
    }
  return dt;  // return dt corresponding to proba_tot
}

void Layer::update ( int n , int s , int s2){ proba_max+=(*(sublayers+s))->update(n,s2); }
```

4.4.3 sublayer.h

```
#ifndef SUBLAYER_H
#define SUBLAYER_H

#include "medium.h"
#include "include.h"
#include "event.h"

class Medium;
class Event;

class Sublayer{
  // Attributes
 private:
  Event **events;
  int smax;
  int column;
 public:
  double proba_tot;
  // Operations
 public:
```

```
    Sublayer (int,int,Medium*);
    ~Sublayer();
    Reac* choose_reac ( double );
    double update ( int n, int s );
};

#endif
```

4.4.4 sublayer.cpp

```
#include "sublayer.h"

Sublayer::Sublayer(int s, int pos, Medium *medium){
    int i;
    smax=s;
    column=pos;
    proba_tot=0;
    events=new Event*[s];
    for (i=0; i<smax; i++) *(events+i)=new Event(medium);
}

Sublayer::~Sublayer(){
    for (int i=0; i<smax; i++) delete *(events+i);
    //free(events);
    delete events;
}

Reac* Sublayer::choose_reac (double alea){
    double tmp2=0, tmp3=0;
    int i=0;
    do
        {
            tmp3=tmp2;
            tmp2+=(*(events+i))->proba_tot;
            i++;
        }
    while(alea>=tmp2 && i<smax);
    if (alea==tmp2)
        {
            cout << "bounding problem in sublayer ! alea="<< alea
                 << " proba_max=" << proba_tot << "\n";
            return NULL;
        }
    return (*(events+i-1))->choose_reac(alea-tmp3);
}

double Sublayer::update ( int n , int s){
    double diff_proba=(*(events+s))->update(n,column,s);
    proba_tot+=diff_proba;
    return diff_proba;
}
```

4.5 Event Class

This class instantiate the several possible reactions than can occur in the system, and can perform them when asked through the medium->layer->sublayer chain.

4.5.1 event.h

```
#ifndef EVENT_H
```

```
#define EVENT_H

#include "include.h"
#include "medium.h"

class Medium;

class Event {
  // Attributes
 private:
  int nr;
  Medium *medium;
  Reac *reac[MAX_REAC_ONE_POINT];
 public:
  double proba_tot;
  // Operations
 public:
  Event(Medium *);
  ~Event();
  Reac* choose_reac ( double alea );
  double update(int,int,int);
 private:
  void update_reac(int,int,int,char,char);
  void change_reac(int, int, int, int, char, char);
};

#endif
```

4.5.2 event.cpp

```
#include "event.h"

Event::Event(Medium *parent){
  nr=0;
  proba_tot=0;
  medium=parent;
  for (int i=0; i<MAX_REAC_ONE_POINT; i++) reac[i]=new Reac();
}

Event::~Event(){
  for (int i=0; i<MAX_REAC_ONE_POINT; i++) delete(reac[i]);
}

void Event::change_reac(int i, int n, int s, int s2, char type, char next){
  reac[i]->n=n;
  reac[i]->s=s;
  reac[i]->s2=s2;
  reac[i]->type=type;
  reac[i]->proba=medium->proba(type);
  reac[i]->next=next;
}

void Event::update_reac(int n, int s, int s2, char root, char catal){
  nr+=2;
  root %= 2;
  catal %= 2;
  if (root == catal) // i.e. root and catal are same config
    {
      change_reac(nr-2,n,s,s2,REAC_11,2-root);    // id root ( & catal)
      change_reac(nr-1,n,s,s2,REAC_13,1+root);    // opp root (& catal)
    }
```

```
    else
      {
        change_reac(nr-2,n,s,s2,REAC_12a,2-root);     // id root
        change_reac(nr-1,n,s,s2,REAC_12b,2-catal);     // id catal
      }
}

double Event::update(int n, int s, int s2){
  int i=0;
  char tmp,tmp2;
  double old_proba=proba_tot;
  tmp=medium->what(n,s,s2);
  if (tmp!=0)
    {
      nr=1;
      change_reac(0,n,s,s2,REAC_20,tmp+2); // Gx-->Ix
    }
  else
    {
      tmp=medium->what(n-1,s,s2);
      nr=2;
      change_reac(0,n,s,s2,REAC_01,tmp-2);
      change_reac(1,n,s,s2,REAC_02,5-tmp);   // L-->G non catal
      tmp2=medium->what(n,s-1,s2);
      if (tmp2!=0)
        update_reac(n,s,s2,tmp,tmp2);          // L--> G catal left
      tmp2=medium->what(n,s+1,s2);
      if (tmp2!=0)
        update_reac(n,s,s2,tmp,tmp2);          // L--> G catal right
      tmp2=medium->what(n,s,s2-1);
      if (tmp2!=0)
        update_reac(n,s,s2,tmp,tmp2);          // L--> G catal front
      tmp2=medium->what(n,s,s2+1);
      if (tmp2!=0)
        update_reac(n,s,s2,tmp,tmp2);          // L--> G catal behind
    }
  proba_tot=0;   //update proba_tot
  for (i=0; i<nr; i++) proba_tot+=reac[i]->proba;
  return proba_tot-old_proba;
}

Reac* Event::choose_reac ( double alea ){
  int i=0;
  double tmp=0;
  do
    {
      tmp+=reac[i]->proba;
      i++;
    }
  while(alea>=tmp && i<MAX_REAC_ONE_POINT);
  if (alea==tmp)
    {
      cout << "bounding problem in event ! alea="<< alea
           << " proba_tot=" << proba_tot << "\n";
      return NULL;
    }
  return reac[i-1];
}
```