

Selected Topics in Programming Assignment

Jens Jacob Torvin Møller

June 14, 2024

Listing 1: ./Benchmark.h

```
1 //
2 // Created by jjtor on 11/06/2024.
3 //
4
5 #ifndef EKSAMENSPROJEKT_BENCHMARKER_H
6 #define EKSAMENSPROJEKT_BENCHMARKER_H
7
8 #include "bits/stdc++.h"
9
10 class Bench {
11 public:
12     void start_clock(std::string name);
13     void stop_clock(std::string name);
14     void report();
15 private:
16     struct tp {
17         std::string name;
18         std::chrono::high_resolution_clock::time_point start;
19         std::chrono::high_resolution_clock::time_point end;
20     };
21     std::vector<tp> storage;
22 };
23
24 #endif //EKSAMENSPROJEKT_BENCHMARKER_H
```

Listing 2: ./GlobalState.h

```
1 //
2 // Created by jjtor on 30/05/2024.
3 //
4
5 #ifndef EKSAMENSPROJEKT_GLOBALSTATE_H
6 #define EKSAMENSPROJEKT_GLOBALSTATE_H
7
8 #include <bits/stdc++.h>
9 #include "Molecule.h"
10 #include "Reaction.h"
11
12
13
14
15
16 #endif //EKSAMENSPROJEKT_GLOBALSTATE_H
```

Listing 3: ./Grapher.h

```
1 //
2 // Created by jjtor on 02/06/2024.
3 //
```

```

4
5 #ifndef EKSAMENSPROJEKT_GRAPHER_H
6 #define EKSAMENSPROJEKT_GRAPHER_H
7
8 #include <bits/stdc++.h>
9
10 #include <utility>
11
12 std::string cleanString(std::string str) {
13     str.erase(std::remove(str.begin(), str.end(), ':'), str.end());
14     return str;
15 }
16
17 class Grapher{
18 private:
19     std::string start = "digraph{";
20     std::string end = "}";
21     std::ofstream out;
22     int delay_index = 0;
23     std::vector<std::string> molecule_labels = std::vector<std::string>();
24     std::vector<double> delays = std::vector<double>();
25 public:
26     Grapher(std::string name){
27         std::string path="..\out\out_" + cleanString(name) + ".txt";
28         out = std::ofstream(path);
29         out << start;
30     };
31     ~Grapher(){
32         out << end;
33         std::cout << "Grapher destroyed. ";
34     }
35
36     void Graph(std::list<stochastic::Reaction> reactions){
37         for (auto r:reactions) {
38             auto delay = AddDelay(r.get_current_rate_parameter());
39             for (auto reactant:r.get_reactants()) {
40                 AddMolecule(reactant.GetName());
41                 for (auto p:r.get_products()) {
42                     AddArrow(reactant.GetName(), p.GetName(), r.get_current_rate_parameter());
43                 }
44             }
45         }
46         std::cout << "Graphing done";
47     }
48
49     void AddMolecule(std::string label){
50         if (std::find(molecule_labels.begin(), molecule_labels.end(), label) !=
51             ↪molecule_labels.end()){
52             return;
53         }
54         molecule_labels.push_back(label);
55         out << label << "[label=\"\" << label <<
56             ↪R\"(", shape="box", style="filled", fillcolor="cyan");\" << "\n";
57     }
58     std::string AddDelay(double delay){
59         /*if (std::find(delays.begin(), delays.end(), delay) != delays.end()){
60             return "r" + std::to_string(delay_index);
61         }*/
62         delays.push_back(delay);
63         auto r = "r" + std::to_string(delay_index);
64         //out << "r" + std::to_string(delay_index) << "[label=\"\" << std::to_string(delay) <<

```

```

↳R(" ",shape="oval",style="filled",fillcolor="yellow"];)" << "\n";
63     delay_index++;
64     return r;
65 }
66
67 //Takes in (label, delay) || (delay, label)
68 template<class T1, class T2>
69 void AddArrow(T1 source, T2 target, double delay){
70     out << source << "->" << target << "[label=\" " << (double)(delay / 0.01) * 0.01 <<"\"]\n";
71 }
72 };
73
74
75 #endif //EKSAMENSPROJEKT_GRAPHER_H

```

Listing 4: ./Molecule.h

```

1 //
2 // Created by jjtor on 06/05/2024.
3 //
4
5 #ifndef EKSAMENSPROJEKT_MOLECULE_H
6 #define EKSAMENSPROJEKT_MOLECULE_H
7 #include <stdlib.h>
8 #include <string>
9
10
11
12
13 #endif //EKSAMENSPROJEKT_MOLECULE_H

```

Listing 5: ./Observer.h

```

1 //
2 // Created by jjtor on 11/06/2024.
3 //
4
5 #ifndef EKSAMENSPROJEKT_OBSERVER_H
6 #define EKSAMENSPROJEKT_OBSERVER_H
7
8 #endif //EKSAMENSPROJEKT_OBSERVER_H

```

Listing 6: ./PrettyPrinter.h

```

1 //
2 // Created by jjtor on 31/05/2024.
3 //
4
5 #ifndef EKSAMENSPROJEKT_PRETTYPRINTER_H
6 #define EKSAMENSPROJEKT_PRETTYPRINTER_H
7
8 #include <bits/stdc++.h>
9 #include "StochasticSimulation.h"
10
11
12
13 /*template<class T>
14 std::ostream& operator<<(std::ostream& os, Molecule molecule){
15     os << molecule.GetName();
16     return os;
17 }*/
18

```

```
19 #endif //EKSAMENSPROJEKT_PRETTYPRINTER_H
```

Listing 7: ./Reaction.h

```
1 //
2 // Created by jjtor on 30/05/2024.
3 //
4
5 #ifndef EKSAMENSPROJEKT_REACTION_H
6 #define EKSAMENSPROJEKT_REACTION_H
7
8 #include <bits/stdc++.h>
9
10
11
12
13
14
15 #endif //EKSAMENSPROJEKT_REACTION_H
```

Listing 8: ./StochasticSimulation.h

```
1 //
2 // Created by jjtor on 06/05/2024.
3 //
4
5 #pragma once
6
7 #include <bits/stdc++.h>
8 #include <chrono>
9 #include <functional>
10 #include <algorithm>
11 #include "Reaction.h"
12 #include "Molecule.h"
13 #include <map>
14
15 namespace stochastic {
16
17     class Environment {
18     public:
19         Environment() {};
20     };
21
22     class Reaction;
23
24     class Molecule {
25     private:
26         std::string symbol;
27         int current_amount;
28     public:
29         Molecule(std::string name, double amount) { symbol = name, current_amount = amount; }
30
31         int get_current_amount() const { return current_amount; }
32
33         std::string GetName() const { return symbol; }
34
35         void set_current_amount(int val) { current_amount = val; }
36
37         //Overloads
38         Reaction operator+(Molecule molecule) const;
39
40 }
```

```

41     Reaction operator+(Reaction reaction);
42
43     Reaction operator>>(double delay) const;
44
45 };
46
47 class GlobalState {
48 private:
49     double time = 0;
50     //std::list<Molecule> reactants; //Current molecules swimming around
51
52     template<class T, class U>
53     struct GenericLookupTable {
54         std::map<T, U> table;
55
56         auto LookUp(T search) {
57             return table.find(search);
58
59             /*if (auto it = table.find(search); it != table.end())
60                 return it;*/
61         }
62
63         void Insert(Molecule m) {
64             table.insert({m.GetName(), m.get_current_amount()});
65         }
66
67         void Update(T element, int value) {
68             auto it = LookUp(element);
69             if (it != table.end()) {
70                 it->second += value;
71             } else {
72                 table.insert({element, value});
73                 std::cout << element << " is not in symbol table. Has now been inserted";
74             }
75         }
76
77     };
78
79     std::list<Reaction> reactions;
80
81 public:
82     GlobalState() {};
83     Environment environment;
84     GenericLookupTable<std::string, int> symbolTable = GenericLookupTable<std::string, int>();
85
86     void AddReactant(Molecule reactant) {
87         //reactants.push_back(reactant);
88         symbolTable.Insert(reactant);
89     }
90
91     void AddTime(double time_to_add) { time += time_to_add; }
92
93     double GetCurrentTime() { return time; }
94 };
95
96 class Reaction {
97 private:
98     std::vector<Molecule> reactants;
99     double rate_parameter;
100    double delay;
101    std::vector<Molecule> products;

```

```

102 public:
103     Reaction() {
104         delay = std::numeric_limits<double>::infinity();
105     };
106
107     double get_current_rate_parameter() const { return rate_parameter; }
108
109     double get_current_delay() const { return delay; }
110
111     void set_delay(double d) { delay = d; }
112
113     void set_rate_parameter(double rp) { rate_parameter = rp; }
114
115     std::vector<Molecule> &get_reactants() { return reactants; }
116
117     std::vector<Molecule> &get_products() { return products; }
118
119     void add_reactant(const Molecule &reactant) { reactants.push_back(reactant); }
120
121     void add_product(const Molecule &product) { products.push_back(product); }
122
123     //Overloads
124     Reaction operator>>(double rate) {
125         auto r = Reaction();
126         for (const auto &reactant: this->get_reactants()) {
127             r.add_reactant(reactant);
128         }
129         r.set_rate_parameter(rate);
130         return r;
131     };
132
133     Reaction operator>=(Molecule molecule) {
134         add_product(molecule);
135         return *this;
136     };
137
138     Reaction operator>=(Reaction reaction) { //TODO: Add copy assignment constructor to ↗
139     ↪reaction to copy all of this, instead of manually doing so
140         auto r = Reaction();
141         for (const auto &reactant: reaction.get_reactants()) {
142             r.add_product(reactant);
143         }
144         for (const auto &reactant: this->get_reactants()) {
145             r.add_reactant(reactant);
146         }
147         r.set_rate_parameter(this->get_current_rate_parameter());
148         return r;
149     };
150
151     Reaction operator>=(Environment env) {
152         ;
153         return *this;
154     };
155
156     class Vessel {
157     private:
158         std::string name;
159         std::list<Reaction> reactions;
160     public:
161         Vessel(std::string n) { name = n; }

```

```

162
163 GlobalState global_state = GlobalState(); //Environment
164 std::list<Reaction> &GetReactions() { return reactions; }
165
166 std::string GetName() { return name; }
167
168 Molecule add(std::string name, double amount) {
169     auto molecule = Molecule(name, amount);
170     global_state.AddReactant(molecule);
171     return molecule;
172 };
173
174 void add(const Reaction reaction) {
175     reactions.push_back(reaction);
176 };
177 };
178
179 stochastic::Reaction FindSmallestDelayReaction(stochastic::Vessel &vessel);
180 class StochasticSimulation {
181 private:
182     std::string path = "..\\out\\trajectory.csv";
183     std::ofstream trajectory;
184 public:
185     StochasticSimulation() {}
186
187     void RunSimulation(Vessel vessel, double end_time);
188
189     template<class Obs>
190     void RunSimulation(Vessel vessel, double end_time, Obs observer){
191         std::string mCount;
192         std::string header;
193         this->path = "..\\out\\trajectory_" + vessel.GetName() + ".csv";
194         trajectory = std::ofstream(path);
195         header += "Time,";
196         trajectory << header;
197         for (auto it = vessel.global_state.symbolTable.table.begin();
198             it != vessel.global_state.symbolTable.table.end(); ++it) {
199             if (std::next(it) != vessel.global_state.symbolTable.table.end()) {
200                 trajectory << it->first << ",";
201             } else {
202                 trajectory << it->first << "\n";
203             }
204         }
205         std::cout << "Running simulation. Time: " + ↵
↵std::to_string(vessel.global_state.GetCurrentTime()) + "\n";
206
207         while (vessel.global_state.GetCurrentTime() <= end_time) {
208             observer(vessel.global_state.GetCurrentTime(), vessel);
209             for (const auto &[key, value]: vessel.global_state.symbolTable.table) {
210                 mCount += std::to_string(value) + ",";
211             }
212             mCount.pop_back();
213             trajectory << vessel.global_state.GetCurrentTime() << ',' << mCount;
214             trajectory << "\n";
215             mCount.clear();
216             for (auto &r: vessel.GetReactions()) {
217                 auto delay = ComputeReactionTime(r, vessel);
218                 r.set_delay(delay);
219                 //Fix so it is taken directly from symbol table without for-range loop
220             }
221

```

```

222         // Pick reaction with shortest delay (reaction time)
223         auto min_delay_reaction = FindSmallestDelayReaction(vessel);
224
225         vessel.global_state.AddTime(min_delay_reaction.get_current_delay()); //Line 5
226         for (auto &q: min_delay_reaction.get_reactants()) {
227             if (std::all_of(min_delay_reaction.get_reactants().begin(), ↵
↵min_delay_reaction.get_reactants().end(),
228                     [&](Molecule &i) {
229                         return ↵
↵vessel.global_state.symbolTable.Lookup(i.GetName())->second > 0;
230                     })) {
231             //TODO: Implement lookup/symbol table (To be..)
232             vessel.global_state.symbolTable.Update(q.GetName(), -1);
233         }
234     }
235     for (auto &p: min_delay_reaction.get_products()) {
236         vessel.global_state.symbolTable.Update(p.GetName(), 1);
237     }
238     std::cout << "Simulation step done. Time: " + ↵
↵std::to_string(vessel.global_state.GetCurrentTime()) + "\n";
239 }
240     std::cout << "Simulation done. Time: " + ↵
↵std::to_string(vessel.global_state.GetCurrentTime());
241 }
242
243 template<class Obs>
244 void RunSimulationParallel(Vessel vessel, double end_time, int numberOfSims, Obs observer);
245 void RunSimulationParallel(Vessel vessel, double end_time, int numberOfSims);
246
247 static double ComputeReactionTime(Reaction &reaction, Vessel &vessel);
248 };
249
250
251 // Pretty printing
252 template<class T>
253 std::ostream &operator<<(std::ostream &os, std::list<T> const &container) {
254     for (auto reaction: container) {
255         os << "Reactants: [";
256         for (auto reactant: reaction.get_reactants()) {
257             os << reactant.GetName() << " ";
258         }
259         os << "\b\b";
260         os << "] Rate parameter: ";
261         os << "[" << reaction.get_current_rate_parameter() << "]";
262         os << " Products: [";
263
264         if (!reaction.get_products().empty()) {
265             for (const auto p: reaction.get_products()) {
266                 os << p.GetName() << " ";
267             }
268             os << "\b\b";
269         }
270         os << "]\n";
271     }
272     return os;
273 }
274 }

```

Listing 9: ./Vessel.h

```

1 //
2 // Created by jjtor on 30/05/2024.

```



```

3  //
4
5  #ifndef EKSAMENSPROJEKT_VESSEL_H
6  #define EKSAMENSPROJEKT_VESSEL_H
7
8  #include "GlobalState.h"
9
10
11
12
13 #endif //EKSAMENSPROJEKT_VESSEL_H

```

Listing 10: ./Benchmark.cpp

```

1  //
2  // Created by jjtor on 14/06/2024.
3  //
4  #include "Benchmark.h"
5  #include <iostream>
6
7  void Bench::start_clock(std::string name) {
8      tp timepoint;
9      timepoint.name = name;
10     timepoint.start = std::chrono::high_resolution_clock::now();
11     storage.push_back(timepoint);
12 }
13 void Bench::stop_clock(std::string name) {
14     for (auto &v : storage){
15         if (v.name == name){
16             v.end = std::chrono::high_resolution_clock::now();
17             return;
18         }
19     }
20 }
21
22 void Bench::report() {
23     for (auto &v : storage){
24         std::chrono::duration<double, std::milli> duration;
25         duration = v.end - v.start;
26         std::cout << v.name << " took " << duration.count() << " ms";
27     }
28 }

```

Listing 11: ./GlobalState.cpp

```

1  //
2  // Created by jjtor on 30/05/2024.
3  //

```

Listing 12: ./Grapher.cpp

```

1  //
2  // Created by jjtor on 02/06/2024.
3  //
4
5  #include "Grapher.h"

```

Listing 13: ./main.cpp

```

1  #include <iostream>
2  #include "Vessel.h"
3  #include "Molecule.h"

```

```

4  #include "StochasticSimulation.h"
5  #include "PrettyPrinter.h"
6  #include "Grapher.h"
7  #include "Benchmark.h"
8
9  void HospitalPeak();
10 stochastic::Vessel circadian_rhythm();
11 stochastic::Vessel seihr(uint32_t N);
12 stochastic::Vessel Figure1_1();
13 stochastic::Vessel Figure1_2();
14 stochastic::Vessel Figure1_3();
15
16
17 int main() {
18     //Vessels
19     auto c = circadian_rhythm();
20     auto s = seihr(10000);
21     auto f1 = Figure1_1();
22     auto f2 = Figure1_2();
23     auto f3 = Figure1_3();
24     //std::cout << c.GetReactions();
25
26     //auto grapher = Grapher("Seihr");
27     //grapher.Graph(r);
28     //grapher.Graph(s.GetReactions());
29
30     auto benchmarker = Bench();
31
32     auto sim = stochastic::StochasticSimulation();
33     //sim.RunSimulation(f3, 2000);
34     //sim.RunSimulation(c, 100);
35     //sim.RunSimulation(s, 100);
36     //HospitalPeak();
37
38 #pragma region Single Run 50 Simulations Benchmarking
39     /*benchmarker.start_clock("SEIHR_SINGLE");
40     for (int i = 0; i < 50; ++i) {
41         std::cout << "\n" << "i = " << i;
42         sim.RunSimulation(s, 100);
43     }
44     benchmarker.stop_clock("SEIHR_SINGLE");
45     std::cout << "\n";
46     benchmarker.report();*/
47 #pragma endregion
48 #pragma region Parallel Run 50 Simulations Benchmarking
49     benchmarker.start_clock("SEIHR_PARALLEL");
50     sim.RunSimulationParallel(s, 100, 50);
51     benchmarker.stop_clock("SEIHR_PARALLEL");
52     benchmarker.report();
53 #pragma endregion
54
55     std::cout << "\nHello, World!" << std::endl;
56
57     return 0;
58 }
59
60 //Vessels
61 stochastic::Vessel circadian_rhythm(){
62     const auto alphaA = 50;
63     const auto alpha_A = 500;
64     const auto alphaR = 0.01;

```

```

65     const auto alpha_R = 50;
66     const auto betaA = 50;
67     const auto betaR = 5;
68     const auto gammaA = 1;
69     const auto gammaR = 1;
70     const auto gammaC = 2;
71     const auto deltaA = 1;
72     const auto deltaR = 0.2;
73     const auto deltaMA = 10;
74     const auto deltaMR = 0.5;
75     const auto thetaA = 50;
76     const auto thetaR = 100;
77
78
79     auto v = stochastic::Vessel{"Circadian Rhythm"};
80
81     const auto env = v.global_state.environment;
82
83     const auto DA = v.add("DA", 1);
84     const auto D_A = v.add("D_A", 0);
85     const auto DR = v.add("DR", 1);
86     const auto D_R = v.add("D_R", 0);
87     const auto MA = v.add("MA", 0);
88     const auto MR = v.add("MR", 0);
89     const auto A = v.add("A", 0);
90     const auto R = v.add("R", 0);
91     const auto C = v.add("C", 0);
92
93
94     v.add((A + DA) >> gammaA >>= D_A);
95     v.add(D_A >> thetaA >>= DA + A);
96     v.add((A + DR) >> gammaR >>= D_R);
97     v.add(D_R >> thetaR >>= DR + A);
98     v.add(D_A >> alpha_A >>= MA + D_A);
99     v.add(DA >> alphaA >>= MA + DA);
100    v.add(D_R >> alpha_R >>= MR + D_R);
101    v.add(DR >> alphaR >>= MR + DR);
102    v.add(MA >> betaA >>= MA + A);
103    v.add(MR >> betaR >>= MR + R);
104    v.add((A + R) >> gammaC >>= C);
105    v.add(C >> deltaA >>= R);
106    v.add(A >> deltaA >>= env);
107    v.add(R >> deltaR >>= env);
108    v.add(MA >> deltaMA >>= env);
109    v.add(MR >> deltaMR >>= env);
110
111    return v;
112 }
113 stochastic::Vessel seihr(uint32_t N) {
114
115     auto v = stochastic::Vessel{"COVID19 SEIHR: " + std::to_string(N)};
116     const auto eps = 0.0009; // initial fraction of infectious
117     const auto I0 = size_t(std::round(eps * N)); // initial infectious
118     const auto E0 = size_t(std::round(eps * N * 15)); // initial exposed
119     const auto S0 = N - I0 - E0; // initial susceptible
120     const auto R0 = 2.4;
121     const auto alpha = 1.0 / 5.1; // incubation rate (E -> I) ~5.1 days
122     const auto gamma = 1.0 / 3.1; // recovery rate (I -> R) ~3.1 days
123     const auto beta = R0 * gamma; // infection/generation rate (S+I -> E+I)
124     const auto P_H = 0.9e-3; // probability of hospitalization
125     const auto kappa = gamma * P_H * (1.0 - P_H); // hospitalization rate (I -> H)

```

```

126     const auto tau = 1.0 / 10.12; // removal rate in hospital (H -> R) ~10.12 days
127     const auto S = v.add("S", S0); // susceptible
128     const auto E = v.add("E", E0); // exposed
129     const auto I = v.add("I", I0); // infectious
130     const auto H = v.add("H", 0); // hospitalized
131     const auto R = v.add("R", 0); // removed/immune (recovered + dead)
132     v.add((S + I) >> beta / N >=> E + I); // susceptible becomes exposed by infectious
133     v.add(E >> alpha >=> I); // exposed becomes infectious
134     v.add(I >> gamma >=> R); // infectious becomes removed
135     v.add(I >> kappa >=> H); // infectious becomes hospitalized
136     v.add(H >> tau >=> R); // hospitalized becomes removed
137     return v;
138 }
139 stochastic::Vessel Figure1_1(){
140     auto v = stochastic::Vessel("Fig1_1");
141     const auto env = v.global_state.environment;
142     const auto A = v.add("A", 100);
143     const auto B = v.add("B", 0);
144     const auto C = v.add("C", 1);
145     const auto lambda = 0.001;
146     v.add((A + C) >> lambda >=> B + C);
147     return v;
148 }
149 stochastic::Vessel Figure1_2(){
150     auto v = stochastic::Vessel("Fig1_2");
151     const auto env = v.global_state.environment;
152     const auto A = v.add("A", 100);
153     const auto B = v.add("B", 0);
154     const auto C = v.add("C", 2);
155     const auto lambda = 0.001;
156     v.add((A + C) >> lambda >=> B + C);
157     return v;
158 }
159 stochastic::Vessel Figure1_3(){
160     auto v = stochastic::Vessel("Fig1_3");
161     const auto env = v.global_state.environment;
162     const auto A = v.add("A", 50);
163     const auto B = v.add("B", 50);
164     const auto C = v.add("C", 1);
165     const auto lambda = 0.001;
166     v.add((A + C) >> lambda >=> B + C);
167     return v;
168 }
169
170
171
172 void HospitalPeak(){
173     auto observer = [](double time, stochastic::Vessel &v){
174         static int max = 0;
175         auto hospitalized = v.global_state.symbolTable.Lookup("H")->second;
176         std::cout << "CURRENT HOSPITALIZED: " << hospitalized << "\n";
177         if (hospitalized > max){
178             max = hospitalized;
179             std::cout << "New Peak Hospitalized: " << max << "\n";
180         }
181     };
182
183     auto NDK = 5822763;
184     auto NNJ = 589755;
185     std::vector<int> pops = {NDK, NNJ};
186     for (auto p:pops) {

```

```

187     auto vessel = seihr(p);
188     auto sim = stochastic::StochasticSimulation();
189     sim.RunSimulation(vessel, 100, observer);
190 }
191
192 }

```

Listing 14: ./Molecule.cpp

```

1  //
2  // Created by jjtor on 06/05/2024.
3  //
4
5  #include "Molecule.h"

```

Listing 15: ./PrettyPrinter.cpp

```

1  //
2  // Created by jjtor on 31/05/2024.
3  //
4
5  #include "PrettyPrinter.h"

```

Listing 16: ./Reaction.cpp

```

1  //
2  // Created by jjtor on 30/05/2024.
3  //
4
5  #include "Reaction.h"
6  #include "Molecule.h"

```

Listing 17: ./StochasticSimulation.cpp

```

1  //
2  // Created by jjtor on 06/05/2024.
3  //
4
5  #include "StochasticSimulation.h"
6  #include "GlobalState.h"
7  #include "Molecule.h"
8  #include "Reaction.h"
9  #include <algorithm>
10
11 //Global variables
12
13 namespace stochastic {
14 //Prototypes
15     Reaction FindSmallestDelayReaction(Vessel &vessel);
16
17     std::string cleanString(std::string str) {
18         str.erase(std::remove(str.begin(), str.end(), '.'), str.end());
19         return str;
20     }
21
22     double RandomNumberGen(double delay) {
23         std::random_device rd;
24         std::mt19937 gen(rd());
25         std::exponential_distribution<double> distribution(delay);
26         double d = distribution(gen);
27         return d;
28     }

```

```

29 void StochasticSimulation::RunSimulation(Vessel vessel, double end_time) {
30     std::string mCount;
31     std::string header;
32     auto vName = vessel.GetName();
33     this->path = "..\\out\\trajectory_" + cleanString(vName) + ".csv";
34     trajectory = std::ofstream(path);
35     header += "Time,";
36     trajectory << header;
37     for (auto it = vessel.global_state.symbolTable.table.begin();
38         it != vessel.global_state.symbolTable.table.end(); ++it) {
39         if (std::next(it) != vessel.global_state.symbolTable.table.end()) {
40             trajectory << it->first << ",";
41         } else {
42             trajectory << it->first << "\n";
43         }
44     }
45     std::cout << "Running simulation. Time: " + ↵
↪std::to_string(vessel.global_state.GetCurrentTime()) + "\n";
46
47     while (vessel.global_state.GetCurrentTime() <= end_time) {
48         for (const auto &[key, value]: vessel.global_state.symbolTable.table) {
49             mCount += std::to_string(value) + ",";
50         }
51         mCount.pop_back();
52         trajectory << vessel.global_state.GetCurrentTime() << ',' << mCount;
53         trajectory << "\n";
54         mCount.clear();
55         for (auto &r: vessel.GetReactions()) {
56             auto delay = ComputeReactionTime(r, vessel);
57             r.set_delay(delay);
58             //Fix so it is taken directly from symbol table without for-range loop
59         }
60
61         // Pick reaction with shortest delay (reaction time)
62         auto min_delay_reaction = FindSmallestDelayReaction(vessel);
63
64         vessel.global_state.AddTime(min_delay_reaction.get_current_delay()); //Line 5
65         bool valid_reaction = std::all_of(min_delay_reaction.get_reactants().begin(), ↵
↪min_delay_reaction.get_reactants().end(),
66             [&](Molecule &i) {
67                 return ↵
↪vessel.global_state.symbolTable.LookUp(i.GetName())->second > 0;
68             });
69         for (auto &q: min_delay_reaction.get_reactants()) {
70             if (valid_reaction) {
71                 vessel.global_state.symbolTable.Update(q.GetName(), -1);
72             }
73         }
74         if (valid_reaction){
75             for (auto &p: min_delay_reaction.get_products()) {
76                 vessel.global_state.symbolTable.Update(p.GetName(), 1);
77             }
78         }
79         std::cout << "Simulation step done. Time: " + ↵
↪std::to_string(vessel.global_state.GetCurrentTime()) + "\n";
80     }
81     std::cout << "Simulation done. Time: " + ↵
↪std::to_string(vessel.global_state.GetCurrentTime()) << "\n";
82
83 }
84

```

```

85     template<typename Obs>
86     void StochasticSimulation::RunSimulationParallel(Vessel vessel, double end_time, int ↗
↪ numberOfSims, Obs observer) {
87         std::vector<std::thread> threads;
88         std::vector<StochasticSimulation> simulations(numberOfSims);
89
90         for (int i = 0; i < numberOfSims; ++i) {
91             threads.emplace_back(&StochasticSimulation::RunSimulation, &simulations[i], ↗
↪ std::ref(vessel), end_time, observer);
92         }
93
94         for (auto &thread: threads) {
95             thread.join();
96         }
97     }
98
99     void StochasticSimulation::RunSimulationParallel(Vessel vessel, double end_time, int ↗
↪ numberOfSims) {
100         std::vector<std::thread> threads;
101         std::vector<StochasticSimulation> simulations(numberOfSims);
102
103         for (int i = 0; i < numberOfSims; ++i) {
104             threads.emplace_back(static_cast<void> (StochasticSimulation::*)(Vessel, ↗
↪ double)>(&StochasticSimulation::RunSimulation), &simulations[i], std::ref(vessel), end_time);
105         }
106
107         for (auto &thread: threads) {
108             thread.join();
109         }
110     }
111
112     double StochasticSimulation::ComputeReactionTime(Reaction &reaction, Vessel &vessel) {
113         double total_amount_of_reactants = 1.0;
114         for (const auto &m: reaction.get_reactants()) {
115             if (vessel.global_state.symbolTable.LookUp(m.GetName())->second <= 0) {
116                 return std::numeric_limits<double>::infinity();
117             }
118             total_amount_of_reactants *= ↗
↪ vessel.global_state.symbolTable.LookUp(m.GetName())->second;
119         }
120         return RandomNumberGen(reaction.get_current_rate_parameter() * total_amount_of_reactants);
121     }
122
123     Reaction FindSmallestDelayReaction(Vessel &vessel) {
124         auto min_delay = std::numeric_limits<double>::infinity();
125         auto min_delay_reaction = Reaction();
126         for (auto &r: vessel.GetReactions()) {
127             if (r.get_current_delay() < min_delay) {
128                 min_delay = r.get_current_delay();
129                 min_delay_reaction = r;
130             }
131         }
132         return min_delay_reaction;
133     }
134
135
136     Reaction Molecule::operator+(Molecule molecule) const {
137         auto r = Reaction();
138         r.add_reactant(molecule);
139         r.add_reactant(*this);
140         return r;

```

```

141     }
142
143
144     Reaction Molecule::operator>>(double rate) const {
145         auto r = Reaction();
146         r.add_reactant(*this);
147         r.set_rate_parameter(rate);
148         return r;
149     }
150
151     Reaction Molecule::operator+(Reaction reaction) {
152         auto r = Reaction();
153         for (auto &reactant: reaction.get_reactants()) {
154             r.add_reactant(reactant);
155         }
156         for (auto &product: reaction.get_products()) {
157             r.add_product(product);
158         }
159         r.set_rate_parameter(reaction.get_current_rate_parameter());
160         return r;
161     };
162 }

```

Listing 18: ./Test/Test.cpp

```

1  //
2  // Created by jjtor on 12/06/2024.
3  //
4
5  #include "doctest.h"
6  #include "../StochasticSimulation.h"
7
8  namespace stochastic{
9      TEST_CASE("Test SymbolTable") {
10         auto symbolTable = GlobalState().symbolTable;
11         SUBCASE("LookUpSuccess"){
12             symbolTable.Update("A", 1);
13             CHECK((symbolTable.LookUp("A")->second == 1));
14         }
15         SUBCASE("LookUpSymbolNotExist"){
16             CHECK((symbolTable.LookUp("A") == symbolTable.table.end()));
17         }
18     }
19
20     TEST_CASE("Pretty Print"){
21         auto r = Reaction();
22         r.add_reactant(Molecule("M1", 1));
23         r.add_reactant(Molecule("M2", 4));
24         r.add_product(Molecule("M3", 0));
25         r.set_rate_parameter(1);
26         Vessel v = Vessel("TestVessel");
27         v.add(r);
28         std::stringstream prettyPrint;
29         prettyPrint << v.GetReactions();
30         CHECK((prettyPrint.str() == "Reactants: [M1 M2 \b \b] Rate parameter: [1] Products: [M3 ↵
↵ \b \b]\n"));
31     }
32 }

```

Listing 19: ./Vessel.cpp

```

1  //

```