sP exam mini project

Henrik van Peet

June 16, 2024

1 My results

1.1 Peaks of SEIHR

For NJ, 589.755, the average peak of H (hospitalized) is 126, based on 100 iterations.
For DK, 5822763, the average peak of H (hospitalized) is xx, based on 100 iterations.

1.2 Plots

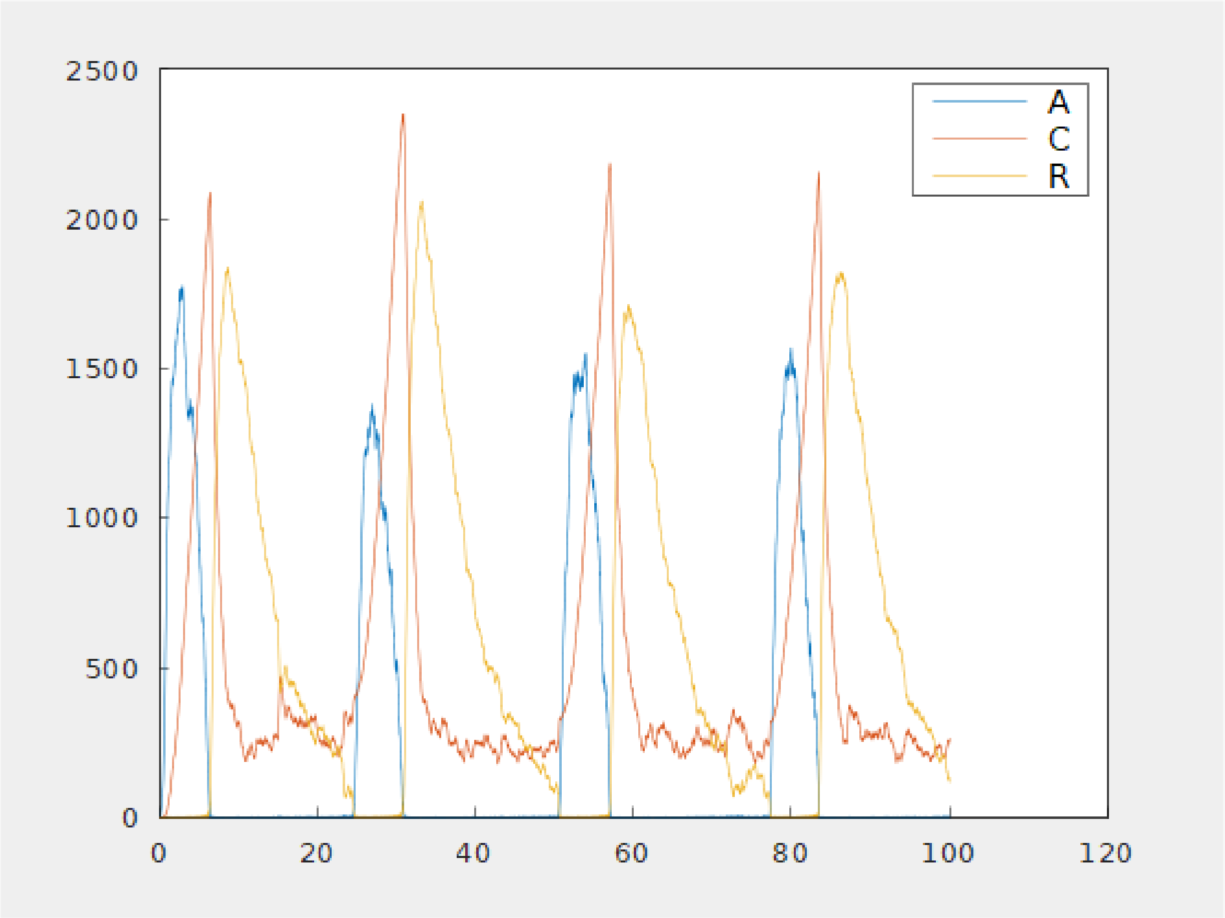


Figure 1: Plot of circadian rhythm

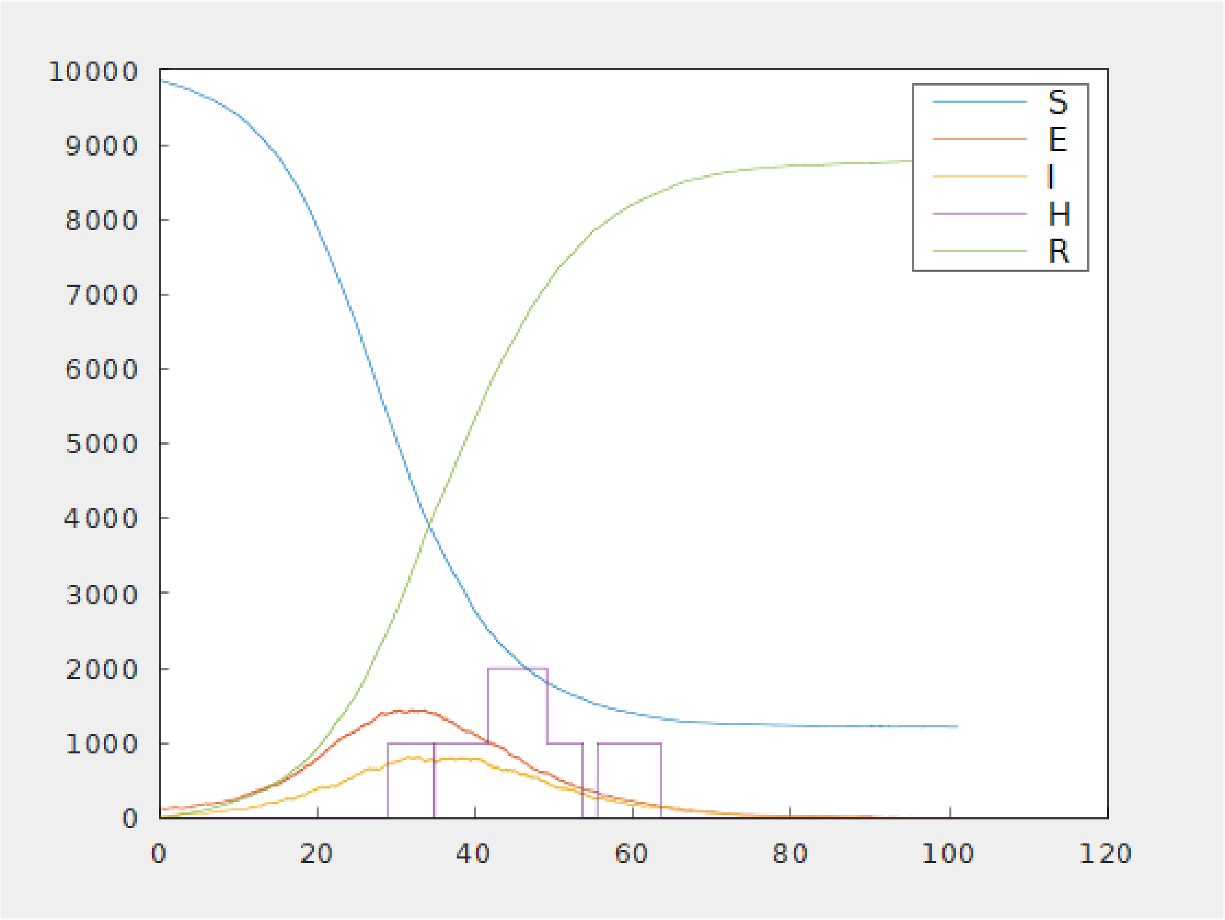


Figure 2: Plot of seihr N=10000

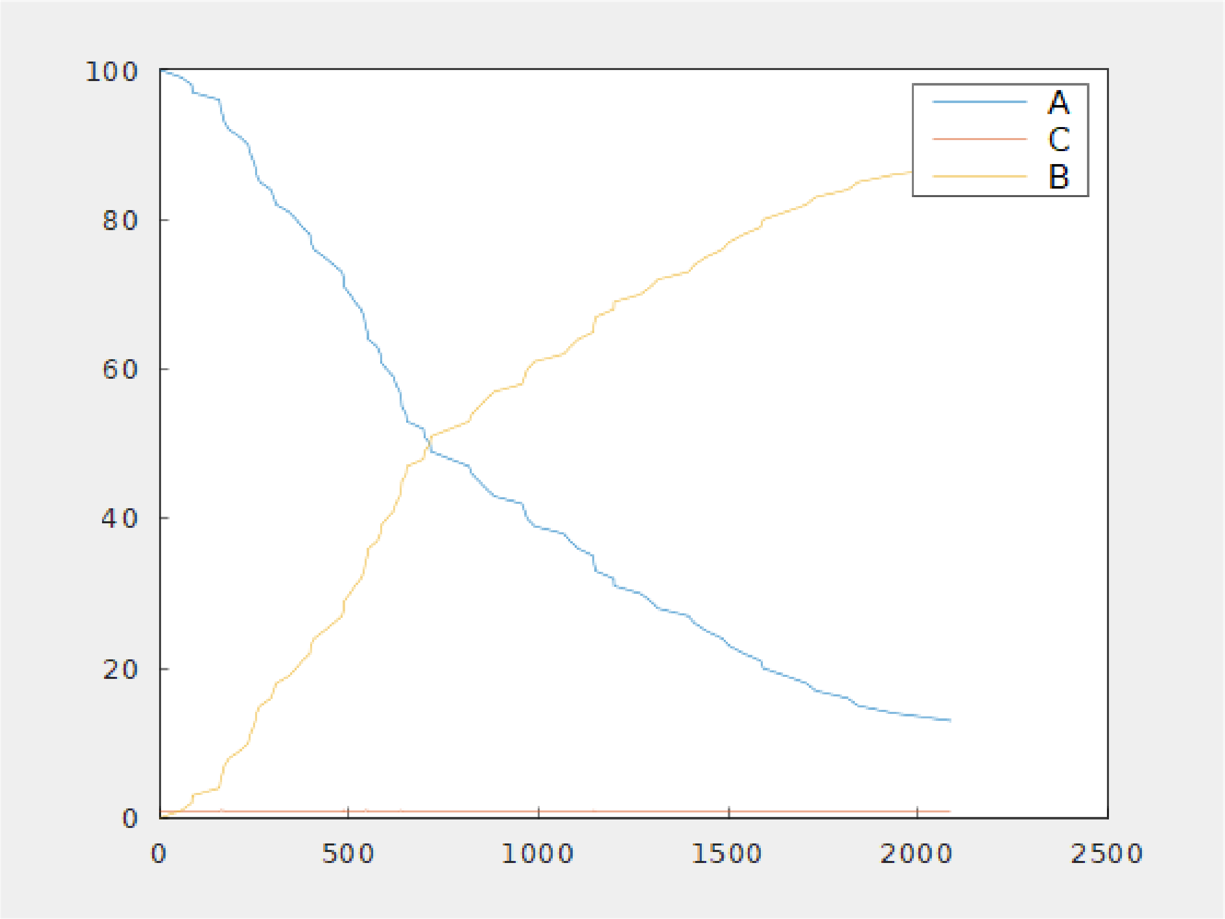


Figure 3: Plot of sample A100 B0 C1

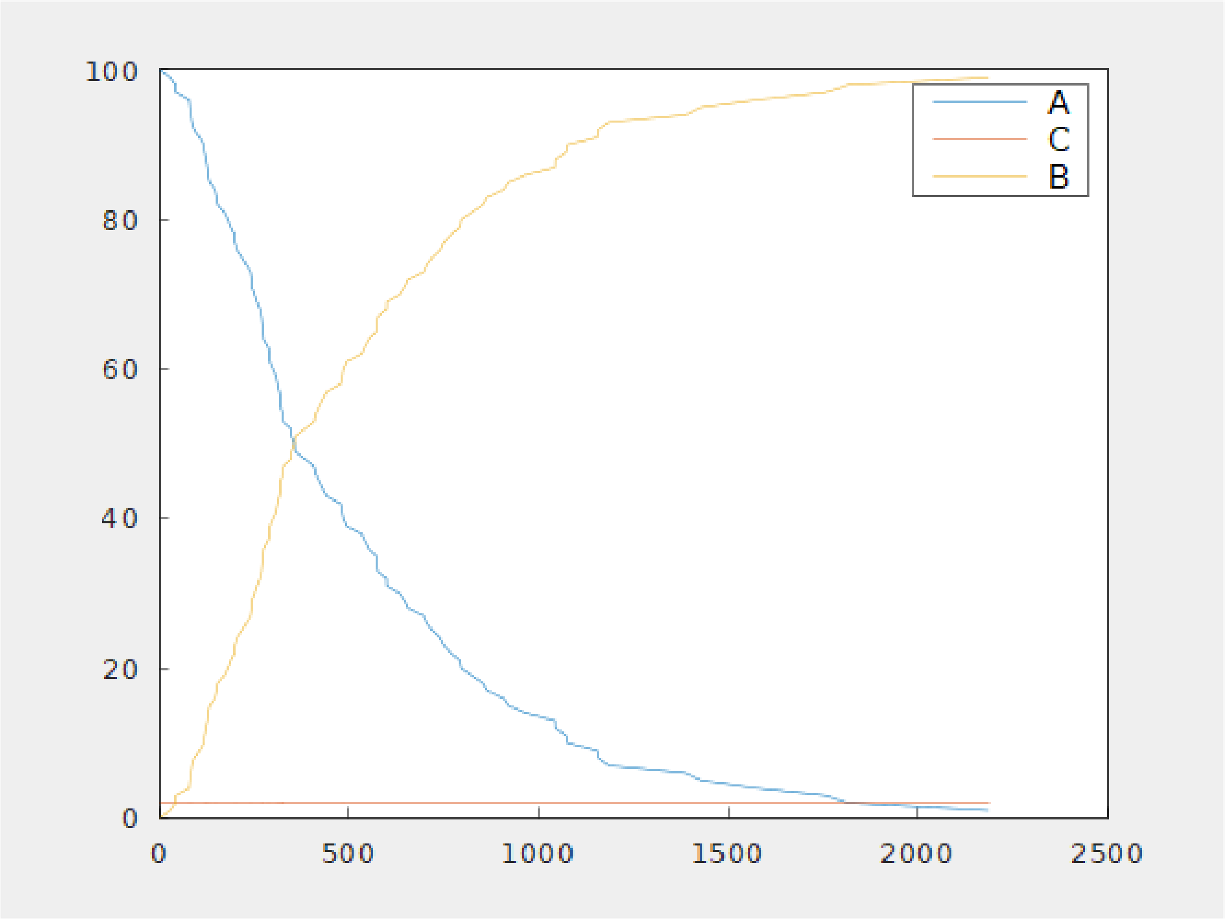
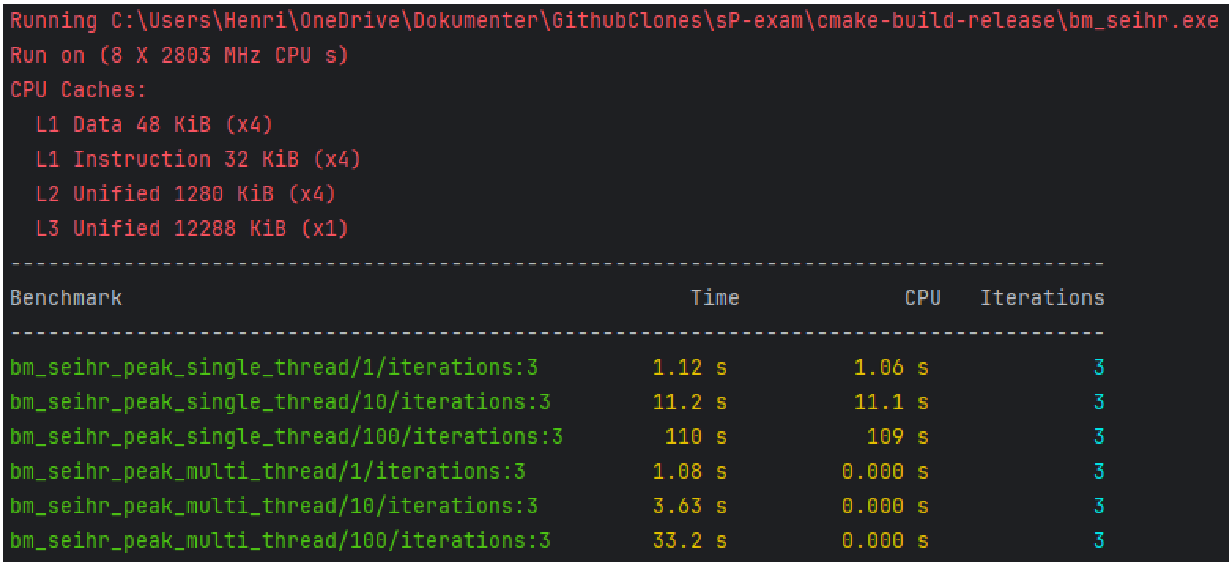


Figure 4: Plot of sample A100 B0 C2

1.3 Benchmark results

Benchmark results from file bm_seihr.cpp.



```
Running C:\Users\Henri\OneDrive\Dokumenter\GithubClones\sP-exam\cmake-build-release\bm_seihr.exe
Run on (8 X 2803 MHz CPU s)
CPU Caches:
  L1 Data 48 KiB (x4)
  L1 Instruction 32 KiB (x4)
  L2 Unified 1280 KiB (x4)
  L3 Unified 12288 KiB (x1)
--------------------------------------------------------------------------------------
Benchmark                                            Time             CPU   Iterations
--------------------------------------------------------------------------------------
bm_seihr_peak_single_thread/1/iterations:3        1.12 s          1.06 s            3
bm_seihr_peak_single_thread/10/iterations:3       11.2 s          11.1 s            3
bm_seihr_peak_single_thread/100/iterations:3       110 s           109 s            3
bm_seihr_peak_multi_thread/1/iterations:3         1.08 s         0.000 s            3
bm_seihr_peak_multi_thread/10/iterations:3        3.63 s         0.000 s            3
bm_seihr_peak_multi_thread/100/iterations:3       33.2 s         0.000 s            3
```

Figure 5: Benchmark results

As can be seen on 5, the running of 100 iterations on a single thread takes approx. 110 seconds. Using the threadpool it only takes 33.2 seconds. It ran 8 threads concurrently.

1.4 Network graph

Sorry, but I could not get you latex code for dot listings to work. The images are rendered using command-line dot utility from Graphiz.

```
digraph{
R[label=R,shape="box",style="filled",fillcolor="cyan"];
C[label=C,shape="box",style="filled",fillcolor="cyan"];
A[label=A,shape="box",style="filled",fillcolor="cyan"];
MR[label=MR,shape="box",style="filled",fillcolor="cyan"];
MA[label=MA,shape="box",style="filled",fillcolor="cyan"];
D_R[label=D_R,shape="box",style="filled",fillcolor="cyan"];
DR[label=DR,shape="box",style="filled",fillcolor="cyan"];
D_A[label=D_A,shape="box",style="filled",fillcolor="cyan"];
DA[label=DA,shape="box",style="filled",fillcolor="cyan"];
r15[label=0.500000,shape="oval",style="filled",fillcolor="yellow"];
MR -> r15
r14[label=10.000000,shape="oval",style="filled",fillcolor="yellow"];
MA -> r14
r13[label=0.200000,shape="oval",style="filled",fillcolor="yellow"];
R -> r13
r0[label=1.000000,shape="oval",style="filled",fillcolor="yellow"];
A -> r0
DA -> r0
r0 -> D_A
r1[label=50.000000,shape="oval",style="filled",fillcolor="yellow"];
D_A -> r1
r1 -> DA
r1 -> A
r2[label=1.000000,shape="oval",style="filled",fillcolor="yellow"];
A -> r2
DR -> r2
r2 -> D_R
r3[label=100.000000,shape="oval",style="filled",fillcolor="yellow"];
D_R -> r3
r3 -> DR
r3 -> A
r4[label=500.000000,shape="oval",style="filled",fillcolor="yellow"];
D_A -> r4
r4 -> MA
r4 -> D_A
r5[label=50.000000,shape="oval",style="filled",fillcolor="yellow"];
DA -> r5
r5 -> MA
r5 -> DA
r6[label=50.000000,shape="oval",style="filled",fillcolor="yellow"];
D_R -> r6
r6 -> MR
r6 -> D_R
r7[label=0.010000,shape="oval",style="filled",fillcolor="yellow"];
DR -> r7
r7 -> MR
r7 -> DR
r8[label=50.000000,shape="oval",style="filled",fillcolor="yellow"];
MA -> r8
r8 -> MA
r8 -> A
r9[label=5.000000,shape="oval",style="filled",fillcolor="yellow"];
MR -> r9
r9 -> MR
r9 -> R
r10[label=2.000000,shape="oval",style="filled",fillcolor="yellow"];
A -> r10
R -> r10
r10 -> C
r11[label=1.000000,shape="oval",style="filled",fillcolor="yellow"];
C -> r11
r11 -> R
r12[label=1.000000,shape="oval",style="filled",fillcolor="yellow"];
A -> r12
}
```

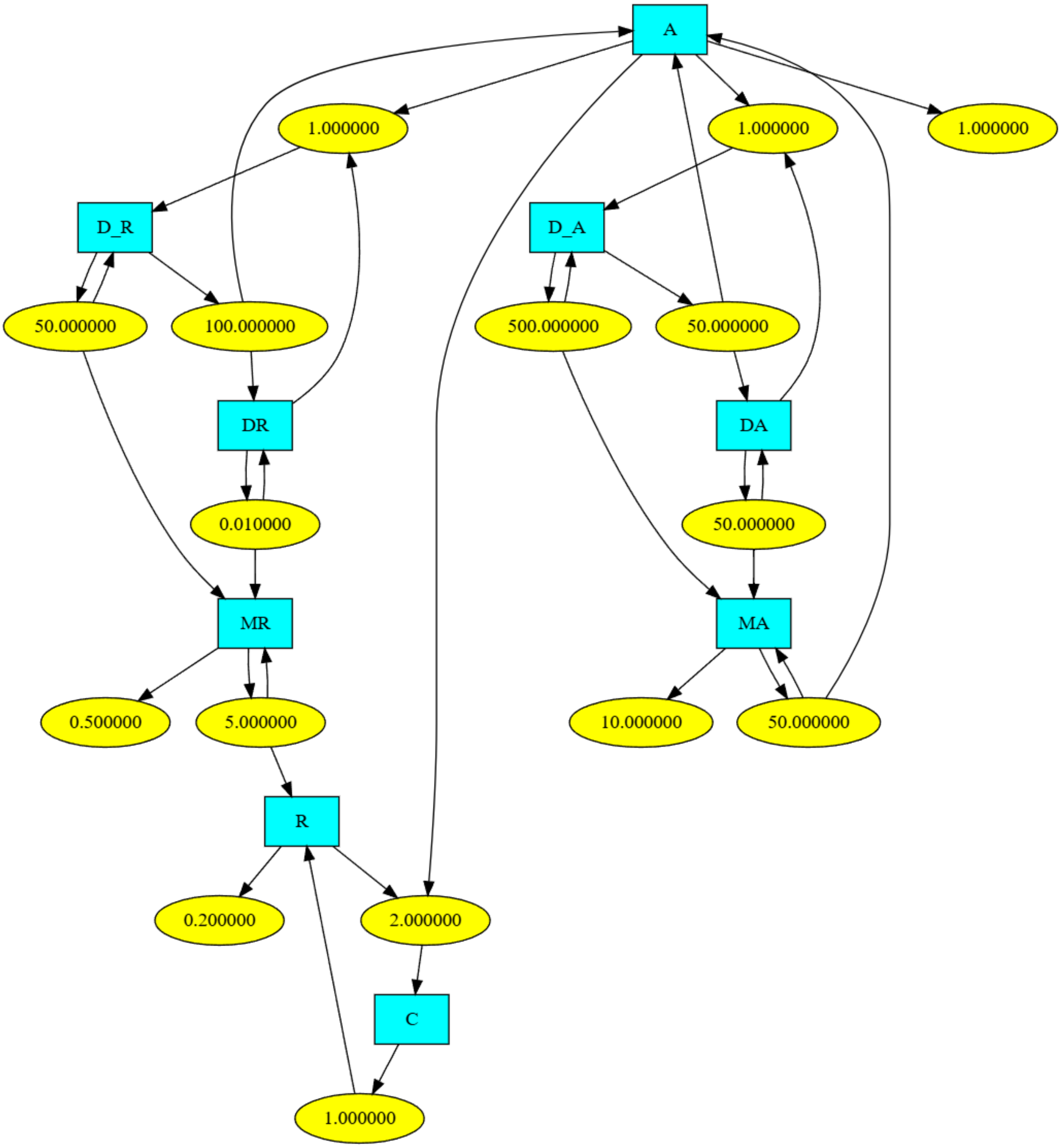


Figure 6: Circadian rhythm network graph

```
digraph{
R[label=R,shape="box",style="filled",fillcolor="cyan"];
H[label=H,shape="box",style="filled",fillcolor="cyan"];
I[label=I,shape="box",style="filled",fillcolor="cyan"];
E[label=E,shape="box",style="filled",fillcolor="cyan"];
S[label=S,shape="box",style="filled",fillcolor="cyan"];
r4[label=0.098814,shape="oval",style="filled",fillcolor="yellow"];
H -> r4
r4 -> R
r3[label=0.000290,shape="oval",style="filled",fillcolor="yellow"];
I -> r3
r3 -> H
r2[label=0.322581,shape="oval",style="filled",fillcolor="yellow"];
I -> r2
r2 -> R
r1[label=0.196078,shape="oval",style="filled",fillcolor="yellow"];
E -> r1
r1 -> I
r0[label=0.000077,shape="oval",style="filled",fillcolor="yellow"];
S -> r0
I -> r0
r0 -> E
r0 -> I
}
```

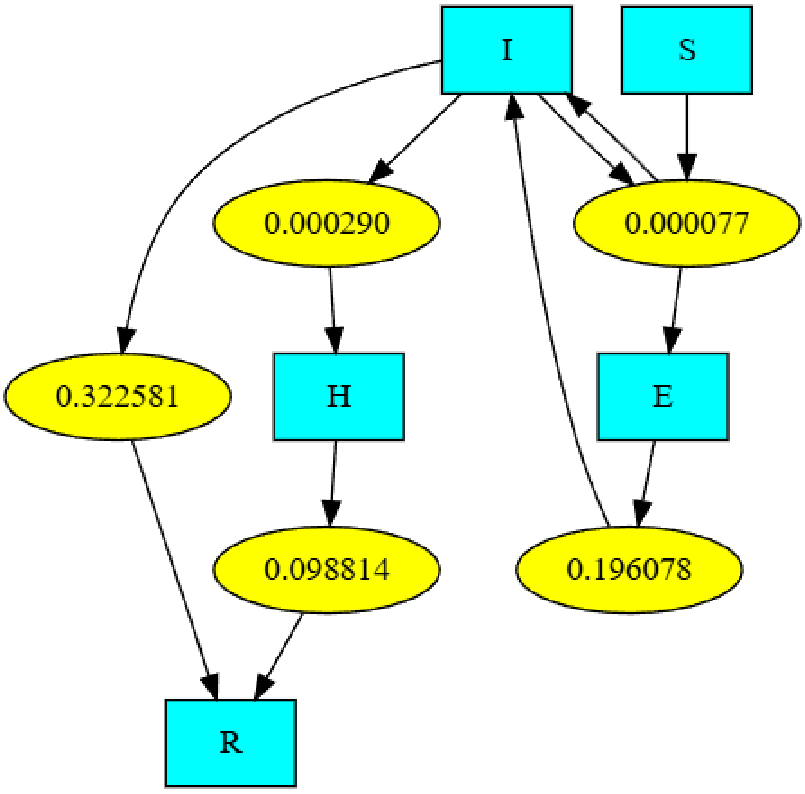


Figure 7: Circadian rhythm network graph

1.5 Compiler

The MinGW compiler is used. The cmake files are included. The cmake used for the benchmarks is the one you used for the corutines exercises.

Listing 1: ./DataPoint.hpp

```
//
// Created by Henri on 28/05/2024.
//

#ifndef SP_EXAM_DATAPOINT_HPP
#define SP_EXAM_DATAPOINT_HPP

#include <string>
```

```cpp
#include <unordered_map>

struct DataPoint{
    explicit DataPoint(double t) : time{t}{}
    double time;
    std::unordered_map<std::string, unsigned> state;
};

#endif //SP_EXAM_DATAPOINT_HPP
```

Listing 2: ./Meta/CanAcceptState.hpp

```cpp
//
// Created by Henri on 28/05/2024.
//

#ifndef SP_EXAM_CANACCEPTSTATE_HPP
#define SP_EXAM_CANACCEPTSTATE_HPP

#include <unordered_map>
#include <string>

template <typename T>
concept CanAcceptState = requires(T&& d, std::unordered_map<std::string, unsigned>& s, const double t) {
    {
    d.accept(s, t) };
};

#endif //SP_EXAM_CANACCEPTSTATE_HPP
```

Listing 3: ./Observers/Observer.hpp

```cpp
//
// Created by Henri on 24/05/2024.
//

#ifndef SHAPE_EXAMPLE_OBSERVER_HPP
#define SHAPE_EXAMPLE_OBSERVER_HPP

#include <unordered_map>
#include <string>
#include <memory>

struct SimulationObserver{
public:
    virtual void accept(const std::unordered_map<std::string, unsigned>& s, double t) = 0;
    virtual ~SimulationObserver() = default;
};

#endif //SHAPE_EXAMPLE_OBSERVER_HPP
```

Listing 4: ./Observers/StateMemorizer.hpp

```cpp
//
// Created by Henri on 12/06/2024.
//

#ifndef SP_EXAM_STATEMEMORIZER_HPP
#define SP_EXAM_STATEMEMORIZER_HPP

#include <vector>
```

```cpp
#include "Observer.hpp"
#include "../DataPoint.hpp"

struct StateMemorizer : public SimulationObserver{
    std::vector<std::string> agentsOfInterest;
    std::vector<DataPoint> data;

    explicit StateMemorizer(std::vector<std::string> agentsOfInterest) : agentsOfInterest(std::move(agentsOfInterest)){}

    void accept(const std::unordered_map<std::string, unsigned>& s, const double t) override{
        DataPoint d{t};
        for (const auto& agentOfInterest: agentsOfInterest) {
            if (!s.contains(agentOfInterest)){
                throw std::logic_error("Could not find: " + agentOfInterest + " in state.");
            }
            auto level = s.at(agentOfInterest);
            d.state.try_emplace(agentOfInterest, level);
        }
        data.push_back(d);
    }
};

#endif //SP_EXAM_STATEMEMORIZER_HPP
```

Listing 5: ./Plotter.hpp

```cpp
//
// Created by Henri on 24/05/2024.
//

#ifndef SP_EXAM_PLOTTER_HPP
#define SP_EXAM_PLOTTER_HPP

#include <string>
#include <vector>
#include <unordered_map>
#include <set>
#include <matplot/matplot.h>


struct Plotter{
    static void visit(const std::vector<DataPoint>& data){
        using namespace matplot;
        std::cout << "Plotting " << data.size() << " data points" << std::endl;
        auto [time, series] = transformData(data);
        hold(on);
        for (const auto& s: series) {
            auto p = plot(time, s.second);
            p->display_name(s.first);
        }
        hold(off);
        legend({});
        show();
    }

    static std::tuple<std::vector<double>, std::unordered_map<std::string, std::vector<unsigned>>> transformData(const std::vector<DataPoint>& data){
        std::vector<double> time;
        std::unordered_map<std::string, std::vector<unsigned>> series;
        for (const auto& [agent, _]: data.front().state) {
            std::vector<unsigned> e;
            series.emplace(agent, e);
        }
        for (const auto& d: data) {
            time.push_back(d.time);
            for (const auto& [agent, level]: d.state) {
                series[agent].push_back(level);
            }
        }
        return {time, series};
    };
};

#endif //SP_EXAM_PLOTTER_HPP
```

Listing 6: ./reactant_store.hpp

```
//
// Created by Henrik on 20-05-2024.
//
#include <unordered_map>
#include <stdexcept>
#include "symbol_table.hpp"

#ifndef SHAPE_EXAMPLE_REACTANT_STORE_H
#define SHAPE_EXAMPLE_REACTANT_STORE_H

template <typename key, typename value> requires std::is_arithmetic_v<value>
class reactant_store : public symbol_table<key, value> {
    void crement(key k, bool increment){
        auto it = this->table.find(k);
        if (it != this->table.end()) {
            if (increment){
                ++(it->second);
            } else {
                --(it->second);
            }
        } else {
            throw std::invalid_argument("The following key does not exist: " + k);
        }
    }

public:
    void increment(key k){
        crement(k, true);
    }

    void decrement(key k){
        crement(k, false);
    }

    std::unordered_map<key, value> getState() const{
        return this->table;
    }
};

#endif //SHAPE_EXAMPLE_REACTANT_STORE_H
```

Listing 7: ./ReactionsToDot.hpp

```
//
// Created by Henri on 28/05/2024.
//
```

```cpp

#ifndef SP_EXAM_REACTIONSTODOT_HPP
#define SP_EXAM_REACTIONSTODOT_HPP

#include <unordered_map>
#include <vector>
#include <utility>
#include <algorithm>
#include <string>
#include <memory>
#include <fstream>
#include <iomanip>
#include <format>
#include "Vessel.hpp"

// Requirement: 2. Provide pretty-printing of the reaction network in ... b) network graph (e.g. Fig. 4).

struct ReactionsToDot{
    static void makeDotFile(const std::unordered_map<unsigned, Reaction>& reactions, const reactant_store<std::string, unsigned>& store, const std::string& filename){
        std::ofstream of{filename};
        of.flags(std::ofstream::fixed);
        of << "digraph{" << std::endl;
        const auto& reactants = store.getState();
        for (const auto& [agentName, _]: reactants) {
            of << agentName;
            addStatement(agentName, "box", "cyan", of);
        }
        for (const auto& [number, reaction]: reactions) {
            std::string reactionID = std::format("r{}", number);
            of << reactionID;
            addStatement(reaction.delay, "oval", "yellow", of);
            for (const auto& agent: reaction.inputs) {
                of << agent << " -> " << reactionID << std::endl;
            }
            for (const auto& agent: reaction.outputs) {
                of << reactionID << " -> " << agent << std::endl;
            }
        }
        of << "}" << std::endl;
    }
private:

    static std::ofstream& addStatement(const std::string& label, const std::string& shape, const std::string& color, std::ofstream& of){
        of << "[label=" << label << R"(,shape=")" << shape << R"(",style="filled",fillcolor=")" << color << "\"];" << std::endl;
        return of;
    }

    static std::ofstream& addStatement(const double& delay, const std::string& shape, const std::string& color, std::ofstream& of){
        of << "[label=" << delay << R"(,shape=")" << shape << R"(",style="filled",fillcolor=")" << color << "\"];" << std::endl;
        return of;
    }
};
#endif //SP_EXAM_REACTIONSTODOT_HPP
```

Listing 8: ./Simulator.hpp

```
//
// Created by Henri on 23/05/2024.
//
#include <memory>
#include <string>
#include <algorithm>
#include <utility>
#include <vector>
#include <memory>
#include <unordered_map>
#include <random>
#include "Meta/CanAcceptState.hpp"
#include "Vessel.hpp"

#ifndef SHAPE_EXAMPLE_SIMULATOR_HPP
#define SHAPE_EXAMPLE_SIMULATOR_HPP

class Simulator{
    std::random_device rd;
    std::mt19937 gen;

public:
    Simulator() : gen(rd()){}
    explicit Simulator(int seed) : gen(seed){}

    // Requirement: 4. Implement the stochastic simulation (Alg. 1) of the system using the reaction rules.
    template <CanAcceptState T>
    void simulate(double duration, Vessel &vessel, T& observer) {
        double t = 0;
        while (t < duration){
            observer.accept(vessel.reactants.getState(), t);
            auto [reactionId, delay] = nextReaction(vessel);
            auto currentReaction = vessel.reactions.find(reactionId)->second;
            t += delay;
            performReaction(vessel, currentReaction);
        }
        observer.accept(vessel.reactants.getState(), t);
    }

private:
    static void performReaction(Vessel& vessel, const Reaction& r) {
        for (auto& input: r.inputs) {
            vessel.reactants.decrement(input);
        }
        for (auto& output: r.outputs) {
            vessel.reactants.increment(output);
        }
    }

    std::tuple<unsigned, double> nextReaction(const Vessel& vessel) {
        unsigned nextReaction = 0;
        double shortestDelay = std::numeric_limits<double>::max();
        bool foundViableReaction = false;
        for (const auto& [index, reaction]: vessel.reactions) {
            auto delay = calculateDelay(reaction, vessel);
            if (delay.has_value()){
                foundViableReaction = true;
                if (delay.value() < shortestDelay){
                    shortestDelay = delay.value();
                    nextReaction = index;
                }
            }
        }
        if (!foundViableReaction){
            throw std::logic_error("No reaction available for simulation.");
        }
        return {nextReaction, shortestDelay};
    }

    std::optional<double> calculateDelay(const Reaction& reaction, const Vessel& vessel){
        double productOfInputs = 1;
        for (const auto& input: reaction.inputs){
            auto inputLevel = vessel.reactants.lookup(input);
            if (inputLevel == 0){
                return std::nullopt;
            }
            productOfInputs *= inputLevel;
        }
        std::exponential_distribution d(productOfInputs * reaction.delay);
        return d(gen);
    }
};

#endif //SHAPE_EXAMPLE_SIMULATOR_HPP
```

Listing 9: ./symbol_table.hpp

```
//
// Created by Henrik on 18-05-2024.
//
#include <unordered_map>
#include <stdexcept>

#ifndef SHAPE_EXAMPLE_SYMBOL_TABLE_H
#define SHAPE_EXAMPLE_SYMBOL_TABLE_H

// 3. Implement a generic symbol reactants to store and lookup objects of user-defined key and value types.
// Support failure cases when
//      a) the reactants does not contain a looked up symbol,
//      b) the reactants already contains a symbol that is being added.

template <typename key, typename value>
class symbol_table{
protected:
    std::unordered_map<key, value> table;
public:
    [[maybe_unused]] bool store(key k, value v){
        auto [_, isSuccess] = table.insert( std::make_pair(k, v));
        return isSuccess;
    }

    value lookup(const key& k) const {
        if (!table.contains(k)){
            throw std::invalid_argument("Table lookup failed for: " + k);
        }
        return table.at(k);
    }
};

#endif //SHAPE_EXAMPLE_SYMBOL_TABLE_H
```

Listing 10: ./Vessel.hpp

```cpp
#include <memory>
#include <string>
#include <algorithm>
#include <utility>
#include <vector>
#include <memory>
#include <unordered_map>
#include <random>
#include "symbol_table.hpp"
#include "reactant_store.hpp"

#ifndef SP_EXAM_VESSEL_HPP
#define SP_EXAM_VESSEL_HPP

struct Environment{
    constexpr static const std::string Name = "Env";
};

struct Agent{
    std::string Name;
    explicit Agent(std::string name) : Name{std::move(name)}{}
};

struct Term{
    std::optional<std::shared_ptr<Term>> LhsTerm;
    std::optional<std::shared_ptr<Agent>> LhsAgent;
    std::optional<std::shared_ptr<Agent>> Rhs;

    Term(const std::shared_ptr<Term>& lhs, const std::shared_ptr<Agent>& rhs) : LhsTerm(lhs), Rhs(rhs) {}

    Term(const std::shared_ptr<Agent>& lhs, const std::shared_ptr<Agent>& rhs) : LhsAgent(lhs), Rhs(rhs){}

    explicit Term(const std::shared_ptr<Agent>& lhs) : LhsAgent(lhs){}

    explicit Term(Environment _){}

    [[nodiscard]] std::vector<std::string> GetAgents() const{
        std::vector<std::string> agents{};
        if (LhsTerm.has_value()){
            auto temp = LhsTerm.value()->GetAgents();
            agents.insert(agents.end(), temp.begin(), temp.end());
        }
        if (LhsAgent.has_value()){
            agents.push_back(LhsAgent.value()->Name);
        }
        if (Rhs.has_value()){
            agents.push_back(Rhs.value()->Name);
        }
        return agents;
    }
};

struct PartialReaction{
    Term Lhs;
    double delay;
    PartialReaction(Term  lhs, double d) : Lhs{std::move(lhs)}, delay{d}{}
};

```

```cpp
struct Reaction{
    std::vector<std::string> inputs{};
    std::vector<std::string> outputs{};
    double delay;

    Reaction(const PartialReaction& p, const Term& rhs) : delay{p.delay} {
        auto lhs = p.Lhs.GetAgents();
        inputs.insert(inputs.end(), lhs.begin(), lhs.end());
        auto rhsAgents = rhs.GetAgents();
        outputs.insert(outputs.end(), rhsAgents.begin(), rhsAgents.end());
    }
};

struct Vessel {
    // Requirement 3. Implement a generic symbol reactants to store and lookup objects of user-defined key and value types.
    // Demonstrate the usage of the symbol reactants with the reactants (names and initial counts).
    // --- The reactans_store is a specialization of the symbol_table.
    reactant_store<std::string, unsigned> reactants{};

    std::string vesselName;
    std::unordered_map<unsigned, Reaction> reactions{};
    unsigned reactionId = 0;


    explicit Vessel(std::string name){
        vesselName = std::move(name);
    }

    static Environment environment(){
        return Environment{};
    }

    std::shared_ptr<Agent> add(const std::string& agentName, unsigned initialAmount){
        auto success = reactants.store(agentName, initialAmount);
        if (!success){
            throw std::invalid_argument("The following key already exists: " + agentName);
        }
        return std::make_shared<Agent>(agentName);
    }

    void add(const Reaction& r){
        reactions.try_emplace(reactionId, r);
        reactionId++;
    }
};

// Requirement: 1. The library should overload operators to support the reaction rule typesetting directly in C++ code.

inline Term operator+(const std::shared_ptr<Agent>& lhs, const std::shared_ptr<Agent>& rhs) {
    return {lhs, rhs};
}

inline std::shared_ptr<Term> operator+(const Environment& env) {
    return std::make_shared<Term>(env);
}

inline PartialReaction operator>>(const std::shared_ptr<Agent>& lhs, const double delay) {
    return {Term(lhs), delay};
}
```

```

inline PartialReaction operator>>(const Term& lhs, const double delay) {
    return {lhs, delay};
}

inline Reaction operator>>=(const PartialReaction& lhs, const Term& rhs) {
    return {lhs, rhs};
}

inline Reaction operator>>=(const PartialReaction& lhs, const std::shared_ptr<Agent>& rhs) {
    return {lhs, Term(rhs)};
}

inline Reaction operator>>=(const PartialReaction& lhs, Environment rhs) {
    return {lhs, Term(rhs)};
}

#endif //SP_EXAM_VESSEL_HPP
```

Listing 11: ./Examples/build_circadian_rhythm.cpp

```
//
// Created by Henri on 24/05/2024.
//

#include "build_circadian_rhythm.h++"

Vessel circadian_rhythm() {
    const auto alphaA = 50;
    const auto alpha_A = 500;
    const auto alphaR = 0.01;
    const auto alpha_R = 50;
    const auto betaA = 50;
    const auto betaR = 5;
    const auto gammaA = 1;
    const auto gammaR = 1;
    const auto gammaC = 2;
    const auto deltaA = 1;
    const auto deltaR = 0.2;
    const auto deltaMA = 10;
    const auto deltaMR = 0.5;
    const auto thetaA = 50;
    const auto thetaR = 100;
    auto v = Vessel{"Circadian Rhythm"};
    const auto env = v.environment();
    const auto DA = v.add("DA", 1);
    const auto D_A = v.add("D_A", 0);
    const auto DR = v.add("DR", 1);
    const auto D_R = v.add("D_R", 0);
    const auto MA = v.add("MA", 0);
    const auto MR = v.add("MR", 0);
    const auto A = v.add("A", 0);
    const auto R = v.add("R", 0);
    const auto C = v.add("C", 0);
    v.add((A + DA) >> gammaA >>= D_A);
    v.add(D_A >> thetaA >>= DA + A);
    v.add((A + DR) >> gammaR >>= D_R);
    v.add(D_R >> thetaR >>= DR + A);
    v.add(D_A >> alpha_A >>= MA + D_A);
    v.add(DA >> alphaA >>= MA + DA);
    v.add(D_R >> alpha_R >>= MR + D_R);
    v.add(DR >> alphaR >>= MR + DR);
    v.add(MA >> betaA >>= MA + A);
    v.add(MR >> betaR >>= MR + R);
    v.add((A + R) >> gammaC >>= C);
    v.add(C >> deltaA >>= R);
    v.add(A >> deltaA >>= env);
    v.add(R >> deltaR >>= env);
    v.add(MA >> deltaMA >>= env);
    v.add(MR >> deltaMR >>= env);
    return v;
}
```

Listing 12: ./Examples/build_sample_trejectory.cpp

```
//
// Created by Henrik on 15-06-2024.
//

#include "build_sample_trejectory.h++"
#include "../Vessel.hpp"

/// Sample trajectory model.
Vessel sample_trajectory(uint32_t A, uint32_t B, uint32_t C)
{
    auto v = Vessel{"Sample trajectories of A, B and C"};
    const auto delay = 0.001; // reaction delay
    const auto agentA = v.add("A", A);
    const auto agentB = v.add("B", B);
    const auto agentC = v.add("C", C);
    v.add((agentA + agentC) >> delay >>= agentB + agentC);

    return v;
}
```

Listing 13: ./Examples/build_seihr.cpp

```
//
// Created by Henri on 12/06/2024.
//

#include "../Vessel.hpp"
#include "build_seihr.h++"

/// SEIHR model for COVID19 epidemic with population size N
Vessel seihr(uint32_t N)
{
    auto v = Vessel{"COVID19 SEIHR: " + std::to_string(N)};
    const auto eps = 0.0009; // initial fraction of infectious
    const auto I0 = size_t(std::round(eps * N)); // initial infectious
    const auto E0 = size_t(std::round(eps * N * 15)); // initial exposed
    const auto S0 = N - I0 - E0; // initial susceptible
    const auto R0 = 2.4; // initial basic reproductive number
    const auto alpha = 1.0 / 5.1; // incubation rate (E -> I) ~5.1 days
    const auto gamma = 1.0 / 3.1; // recovery rate (I -> R) ~3.1 days
    const auto beta = R0 * gamma; // infection/generation rate (S+I -> E+I)
    const auto P_H = 0.9e-3; // probability of hospitalization
    const auto kappa = gamma * P_H * (1.0 - P_H); // hospitalization rate (I -> H)
    const auto tau = 1.0 / 10.12; // removal rate in hospital (H -> R) ~10.12 days
    const auto S = v.add("S", S0); // susceptible
    const auto E = v.add("E", E0); // exposed
    const auto I = v.add("I", I0); // infectious
    const auto H = v.add("H", 0); // hospitalized
    const auto R = v.add("R", 0); // removed/immune (recovered + dead)
    v.add((S + I) >> beta / N >>= E + I); // susceptible becomes exposed by infectious
    v.add(E >> alpha >>= I); // exposed becomes infectious
    v.add(I >> gamma >>= R); // infectious becomes removed
    v.add(I >> kappa >>= H); // infectious becomes hospitalized
    v.add(H >> tau >>= R); // hospitalized becomes removed
    return v;
}
```

Listing 14: ./Executables/bm_seihr.cpp

```
//
// Created by Henri on 14/06/2024.
//

#include <benchmark/benchmark.h>
#include "../utilities.h++"

// 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.
size_t SimulateSeihrPeak_ThreadPool(size_t N, size_t iterations)
{
    vector<unsigned> results{};

    // Create a thread pools
    ThreadPool pool(SimulateSeihrPeak, N, results, iterations);

    pool.waitForCompletion();

    // average of the vector elements
    return average(results);
}


static void bm_seihr_peak_single_thread(benchmark::State& state){
    const auto iterations = state.range(0);
    for (auto _ : state) {
        for (int i = 0; i < iterations; ++i) {
            auto result = SimulateSeihrPeak(COVID19Parameters::NJ_Population);
            benchmark::DoNotOptimize(result);
            benchmark::ClobberMemory();
        }
    }
}

static void bm_seihr_peak_multi_thread(benchmark::State& state){
    const auto iterations = state.range(0);
    for (auto _ : state) {
        auto result = SimulateSeihrPeak_ThreadPool(COVID19Parameters::NJ_Population, iterations);
        benchmark::DoNotOptimize(result);
        benchmark::ClobberMemory();
    }
}

BENCHMARK(bm_seihr_peak_single_thread)->Unit(benchmark::kSecond)->RangeMultiplier(10)->Range(1, 100)->Iterations(1);
//BENCHMARK(bm_seihr_peak_single_thread)->Unit(benchmark::kSecond)->Arg(COVID19Parameters::DK_Population)->It


BENCHMARK(bm_seihr_peak_multi_thread)->Unit(benchmark::kSecond)->RangeMultiplier(10)->Range(1, 100)->Iterations(1);
```

Listing 15: ./Executables/circadian_plot_graph.cpp

```cpp

#include <iostream>
#include "../Vessel.hpp"
#include "../Simulator.hpp"
#include "../Observers/Observer.hpp"
#include "../Examples/build_circadian_rhythm.h++"
#include "../Observers/StateMemorizer.hpp"
#include "../Plotter.hpp"

// Requirement: 5. Demonstrate the application of the library on the three examples (shown in Fig. 2).
// Requirement: 6. Display simulation trajectories of how the amounts change. External tools/libraries can be used to visualize.


int main() {
    auto v = circadian_rhythm();
    std::cout << "Finished building vessel " << v.vesselName << std::endl;

    Simulator sim{1};
    std::vector<std::string> agentsOfInterest = {"A", "C", "R"};
    StateMemorizer memorizer{agentsOfInterest};
    sim.simulate(100.0, v, memorizer);
    Plotter::visit(memorizer.data);
    return 0;
}
```

Listing 16: ./Executables/make_dot_file.cpp

```cpp
//
// Created by Henri on 12/06/2024.
//

#include <iostream>
#include "../Vessel.hpp"
#include "../ReactionsToDot.hpp"
#include "../Examples/build_circadian_rhythm.h++"
#include "../Examples/build_seihr.h++"

int main() {
    auto circadianVessel = circadian_rhythm();
    std::cout << "Finished building vessel:" << circadianVessel.vesselName << std::endl;
    ReactionsToDot::makeDotFile(circadianVessel.reactions, circadianVessel.reactants, "circadian_dot.txt");

    auto seihrVessel = seihr();
    std::cout << "Finished building vessel:" << seihrVessel.vesselName << std::endl;
    ReactionsToDot::makeDotFile(seihrVessel.reactions, seihrVessel.reactants, "seihr_dot.txt");

    return 0;
}
```

Listing 17: ./Executables/run_pretty_printing.cpp

```cpp
//
// Created by Henri on 24/05/2024.
//

```

```
#include <iostream>
#include "../Vessel.hpp"
#include "../Examples/build_circadian_rhythm.h++"
#include "../Examples/build_seihr.h++"
#include "../pretty_printing.h++"

int main() {
    auto v = circadian_rhythm();
    std::cout << "Finished building vessel:" << v.vesselName << std::endl;
    std::cout << "Printing reactions of vessel:" << std::endl;
    std::cout << v;

    v = seihr();
    std::cout << "Finished building vessel:" << v.vesselName << std::endl;
    std::cout << "Printing reactions of vessel:" << std::endl;
    std::cout << v;

    return 0;
}
```

Listing 18: ./Executables/sample_trajectories.cpp

```
//
// Created by Henrik on 15-06-2024.
//


#include <iostream>
#include "../Examples/build_sample_trejectory.h++"
#include "../Simulator.hpp"
#include "../Observers/StateMemorizer.hpp"
#include "../Plotter.hpp"

// Requirement: 5. Demonstrate the application of the library on the three examples (shown in Fig. 1).
// Requirement: 6. Display simulation trajectories of how the amounts change. External tools/libraries can be used to visualize.


int main() {
    auto v = sample_trajectory(100, 0, 1);
    std::cout << "Finished building vessel " << v.vesselName << std::endl;

    Simulator sim{1};

    std::vector<std::string> agentsOfInterest = {"A", "B", "C"};
    StateMemorizer memorizer{agentsOfInterest};
    sim.simulate(2000.0, v, memorizer);
    Plotter::visit(memorizer.data);

    return 0;
}
```

Listing 19: ./Executables/seihr_peak_single_thread.cpp

```
//
// Created by Henri on 12/06/2024.
//

#include <iostream>
#include "../Vessel.hpp"
#include "../Simulator.hpp"
```

```cpp
#include "../utilities.h++"

int main() {
    auto peak = SimulateSeihrPeak(COVID19Parameters::NJ_Population);
    std::cout << "Peak level: " <<  peak  << std::endl;

    return 0;
}
```

Listing 20: ./Executables/seihr_peak_thread_pool.cpp

```cpp
//
// Created by Henri on 13/06/2024.
//
// C++ Program to demonstrate thread pooling
#include <iostream>
#include "../thread_pool.h++"
#include "../utilities.h++"

using namespace std;

// 8. Implement support for multiple computer cores by parallelizing the computation of several simulations at the same time.
// Estimate the likely (average) value of the hospitalized peak over 100 simulations.

int main()
{
    vector<unsigned> results{};

    // Create a thread pool with 4 threads
    ThreadPool pool(SimulateSeihrPeak, COVID19Parameters::DK_Population, results, 100);

    pool.waitForCompletion();

    // average of the vector elements
    auto avg = average(results);
    cout << "Average: " << avg << endl;

    return 0;
}
```

Listing 21: ./Executables/seihr_plot_graph.cpp

```cpp
//
// Created by Henri on 12/06/2024.
//

#include <iostream>
#include "../Vessel.hpp"
#include "../Simulator.hpp"
#include "../Observers/Observer.hpp"
#include "../Observers/StateMemorizer.hpp"
#include "../Examples/build_seihr.h++"
#include "../Observers/PeakObserver.h++"
#include "../Plotter.hpp"

// Requirement: 5. Demonstrate the application of the library on the three examples (shown in Fig. 3).
// Requirement: 6. Display simulation trajectories of how the amounts change. External tools/libraries can be used to visualize.

int main() {
    auto v = seihr();
    std::cout << "Finished building vessel " << v.vesselName << std::endl;

    Simulator sim{1};

    std::vector<std::string> agentsOfInterest = {"S", "E", "I", "H", "R"};
    //std::vector<std::string> agentsOfInterest = {"H"};
    StateMemorizer memorizer{agentsOfInterest};
    sim.simulate(100.0, v, memorizer);
    std::vector<DataPoint> transformedData{};
    std::ranges::for_each(memorizer.data.begin(), memorizer.data.end(), [](DataPoint& d){
        auto& value = d.state.find("H")->second;
        value *= 1000;
    });
    Plotter::visit(memorizer.data);

    return 0;
}
```

Listing 22: ./Tests/pretty_printing_test.cpp

```
//
// Created by Henrik on 15-06-2024.
//
#include "doctest/doctest.h"
#include "../Examples/build_circadian_rhythm.h++"
#include "../pretty_printing.h++"
#include <sstream>

// Requirement: 9. Implement unit tests (e.g. pretty-printing of reaction rules)

TEST_CASE("testing pretty printing of reaction rules")
{
    auto circadianVessel = circadian_rhythm();
    std::stringstream ss;
    ss << circadianVessel;
    CHECK(ss.str() == "Circadian Rhythm\n"
                      "MR--0.5->Env\n"
                      "MA--10->Env\n"
                      "R--0.2->Env\n"
                      "A+DA--1->D_A\n"
                      "D_A--50->DA+A\n"
                      "A+DR--1->D_R\n"
                      "D_R--100->DR+A\n"
                      "D_A--500->MA+D_A\n"
                      "DA--50->MA+DA\n"
                      "D_R--50->MR+D_R\n"
                      "DR--0.01->MR+DR\n"
                      "MA--50->MA+A\n"
                      "MR--5->MR+R\n"
                      "A+R--2->C\n"
                      "C--1->R\n"
                      "A--1->Env\n");
}
```

Listing 23: ./Tests/reactant_store_test.cpp

```
//
// Created by Henrik on 20-05-2024.
//

#include "doctest/doctest.h"
```

```cpp
#include "../reactant_store.hpp"

TEST_CASE("reactant-store increment test (int, int)")
{
    reactant_store<int, int> table{};
    auto key = 1;
    auto value = 42;

    table.store(key, value);
    auto result1 = table.lookup(key);
    CHECK(result1);
    CHECK((value == result1));

    table.increment(key);
    auto result2 = table.lookup(key);
    CHECK((value + 1 == result2));
}

TEST_CASE("reactant-store decrement test (int, int)")
{
    reactant_store<int, int> table{};
    auto key = 1;
    auto value = 42;

    table.store(key, value);
    auto result1 = table.lookup(key);
    CHECK(result1);
    CHECK((value == result1));

    table.decrement(key);
    auto result2 = table.lookup(key);
    CHECK((value - 1 == result2));
}
```

Listing 24: ./Tests/symbol_table_test.cpp

```cpp
//
// Created by Henrik on 18-05-2024.
//

#include "doctest/doctest.h"
#include <string>
#include "../symbol_table.hpp"

// Requirement: 9. Implement unit tests (e.g. test symbol table methods, their failure cases)

TEST_CASE("Symbol table add tests (string, unsigned)")
{
    symbol_table<std::string, unsigned> table{};
    auto key = "key";
    auto otherKey = "otherKey";
    auto value = 1U;

    bool success1 = table.store(key, value);
    CHECK(success1);

    bool success2 = table.store(otherKey, value);
    CHECK(success2);

    bool success3 = table.store(key, value);
    CHECK(!success3);
}
```

```

TEST_CASE("Symbol table look up tests (string, unsigned)")
{
    symbol_table<std::string, unsigned> table{};
    auto key = "key";
    auto otherKey = "otherKey";
    auto value = 1U;

    table.store(key, value);
    auto result1 = table.lookup(key);
    CHECK((result1 == value));
    CHECK_THROWS_AS(table.lookup(otherKey), std::invalid_argument);
}

TEST_CASE("Symbol table add tests (int, int)")
{
    symbol_table<int, int> table{};
    auto key = 1;
    auto otherKey = 2;
    auto value = 42;

    bool success1 = table.store(key, value);
    CHECK(success1);

    bool success2 = table.store(otherKey, value);
    CHECK(success2);

    bool success3 = table.store(key, value);
    CHECK(!success3);
}

TEST_CASE("Symbol table look up tests (int, int)")
{
    symbol_table<int, int> table{};
    auto key = 1;
    auto otherKey = 2;
    auto value = 42;

    table.store(key, value);
    auto result1 = table.lookup(key);
    CHECK((result1 == value));
    CHECK_THROWS_AS(table.lookup(otherKey), std::invalid_argument);
}
```

Listing 25: ./Tests/vessel_test.cpp

```
// Test for the ctor and functionality of the vessel example

#include "doctest/doctest.h"
#include "../Vessel.hpp"

TEST_CASE("Vessel ctor test")
{
    auto v = Vessel{"Circadian Rhythm"};
    CHECK(v.reactions.empty());
    CHECK((v.vesselName == "Circadian Rhythm"));
}

TEST_CASE("Vessel successful parse")
{
    const auto gammaA = 1;
    const auto thetaA = 50;
    auto v = Vessel{"Circadian Rhythm"};
    const auto env = Vessel::environment();
    const auto DA = v.add("DA", 1);
    const auto D_A = v.add("D_A", 0);
    const auto A = v.add("A", 0);
    const auto agentMultiOverload = (A + DA);
    const auto agentAndDelayOverload = (D_A) >> thetaA;
    const auto reaction = (A + DA) >> gammaA >>= D_A;
    v.add((A + DA) >> gammaA >>= D_A);
    v.add(A >> gammaA >>= D_A);
    v.add(A >> gammaA >>= env);
}
```

Listing 26: ./CMakeLists.txt

```
cmake_minimum_required(VERSION 3.27)
project(sP-exam)

set(CMAKE_CXX_STANDARD 23)
include(Cmake/doctest.cmake)
include(Cmake/matplotplusplus.cmake)
include(Cmake/benchmark.cmake)
find_package(Threads REQUIRED)


add_executable(circadian_graph Executables/circadian_plot_graph.cpp
        Vessel.hpp
        symbol_table.hpp
        reactant_store.hpp
        Simulator.hpp
        Observers/Observer.hpp
        Plotter.hpp
        Examples/build_circadian_rhythm.h++
        Examples/build_circadian_rhythm.cpp
        Examples/build_seihr.h++
        Examples/build_seihr.cpp

)
target_link_libraries(circadian_graph PRIVATE doctest::doctest_with_main)
target_link_libraries(circadian_graph PUBLIC matplot)

add_executable(pretty_printing Executables/run_pretty_printing.cpp
        Vessel.hpp
        symbol_table.hpp
        reactant_store.hpp
        Examples/build_circadian_rhythm.h++
        Examples/build_circadian_rhythm.cpp
        Examples/build_seihr.h++
        Examples/build_seihr.cpp
        pretty_printing.h++
)

add_executable(make_dot_file Executables/make_dot_file.cpp
        Vessel.hpp
        symbol_table.hpp
        reactant_store.hpp
        ReactionsToDot.hpp
        Examples/build_circadian_rhythm.h++
        Examples/build_circadian_rhythm.cpp
        Examples/build_seihr.h++
        Examples/build_seihr.cpp
)
```

```
enable_testing()
add_executable(my_test Tests/vessel_test.cpp
        Vessel.hpp
        symbol_table.hpp
        Tests/symbol_table_test.cpp
        reactant_store.hpp
        Tests/reactant_store_test.cpp
        Simulator.hpp
        Observers/Observer.hpp
        ReactionsToDot.hpp
        Examples/build_circadian_rhythm.h++
        Examples/build_circadian_rhythm.cpp
        Examples/build_seihr.h++
        Examples/build_seihr.cpp
        Tests/pretty_printing_test.cpp
        Tests/pretty_printing_test.cpp
        pretty_printing.h++
)

target_link_libraries(my_test PRIVATE doctest::doctest_with_main)
#target_link_libraries(my_test PUBLIC matplot)
add_test(NAME my_test COMMAND my_test)

add_executable(seihr_peak_single_thread Executables/seihr_peak_single_thread.cpp
        Vessel.hpp
        symbol_table.hpp
        reactant_store.hpp
        Simulator.hpp
        Observers/Observer.hpp
        Plotter.hpp
        Examples/build_seihr.h++
        Examples/build_seihr.cpp
        utilities.h++
)
target_link_libraries(seihr_peak_single_thread PUBLIC matplot)

add_executable(seihr_plot_graph Executables/seihr_plot_graph.cpp
        Vessel.hpp
        symbol_table.hpp
        reactant_store.hpp
        Simulator.hpp
        Observers/Observer.hpp
        Plotter.hpp
        Examples/build_seihr.h++
        Examples/build_seihr.cpp
)
target_link_libraries(seihr_plot_graph PUBLIC matplot)

add_executable(seihr_peak_thread_pool Executables/seihr_peak_thread_pool.cpp
        Vessel.hpp
        symbol_table.hpp
        reactant_store.hpp
        Simulator.hpp
        Observers/Observer.hpp
        Plotter.hpp
        Examples/build_seihr.h++
        Examples/build_seihr.cpp
        utilities.h++
)
```

```
add_executable(bm_seihr Executables/bm_seihr.cpp
        Vessel.hpp
        symbol_table.hpp
        reactant_store.hpp
        Simulator.hpp
        Observers/Observer.hpp
        Plotter.hpp
        Examples/build_seihr.h++
        Examples/build_seihr.cpp
        utilities.h++
)

target_link_libraries(bm_seihr PRIVATE Threads::Threads benchmark::benchmark_main)
#add_test(NAME bm_seihr_single_thread COMMAND bm_seihr_single_thread)

add_executable(sample_trajectories_graph Executables/sample_trajectories.cpp
        Examples/build_sample_trejectory.h++
        Examples/build_sample_trejectory.cpp
)
target_link_libraries(sample_trajectories_graph PUBLIC matplot)
```

Listing 27: ./benchmark.txt

```
# Downloads and compiles Google Benchmark
include(FetchContent)
set(FETCHCONTENT_QUIET ON)
set(FETCHCONTENT_UPDATES_DISCONNECTED ON)

set(BENCHMARK_ENABLE_TESTING OFF CACHE BOOL "Enable testing of the benchmark library.")
set(BENCHMARK_ENABLE_EXCEPTIONS ON CACHE BOOL "Enable the use of exceptions in the benchmark library.")
set(BENCHMARK_ENABLE_LTO OFF CACHE BOOL "Enable link time optimisation of the benchmark library.")
set(BENCHMARK_USE_LIBCXX OFF CACHE BOOL "Build and test using libc++ as the standard library.")
set(BENCHMARK_ENABLE_WERROR OFF CACHE BOOL "Build Release candidates with -Werror.")
set(BENCHMARK_FORCE_WERROR OFF CACHE BOOL "Build Release candidates with -Werror regardless of compiler issues.")
set(BENCHMARK_ENABLE_INSTALL OFF CACHE BOOL "Enable installation of benchmark. (Projects embedding benchmark may want to turn this OFF.)")
set(BENCHMARK_ENABLE_DOXYGEN OFF CACHE BOOL "Build documentation with Doxygen.")
set(BENCHMARK_INSTALL_DOCS OFF CACHE BOOL "Enable installation of documentation.")
set(BENCHMARK_DOWNLOAD_DEPENDENCIES ON CACHE BOOL "Allow the downloading and in-tree building of unmet dependencies")
set(BENCHMARK_ENABLE_GTEST_TESTS OFF CACHE BOOL "Enable building the unit tests which depend on gtest")
set(BENCHMARK_USE_BUNDLED_GTEST OFF CACHE BOOL "Use bundled GoogleTest. If disabled, the find_package(GTest) will be used.")
FetchContent_Declare(googlebenchmark
        GIT_REPOSITORY https://github.com/google/benchmark.git
        GIT_TAG v1.8.3    # or "main" for latest
        GIT_SHALLOW TRUE  # download specific revision only (git clone --depth 1)
        GIT_PROGRESS TRUE # show download progress in Ninja
        USES_TERMINAL_DOWNLOAD TRUE)
FetchContent_MakeAvailable(googlebenchmark)

message(STATUS "!!! Benchmark comparison requires python3 and ‘pip install scipy‘ !!!")
set(benchmark_cmp python3 ${googlebenchmark_SOURCE_DIR}/tools/compare.py)
```

Listing 28: ./doctest.txt

```
# Downloads and compiles DocTest unit testing framework
include(FetchContent)
set(FETCHCONTENT_QUIET ON)
set(FETCHCONTENT_UPDATES_DISCONNECTED ON)

set(DOCTEST_WITH_TESTS OFF CACHE BOOL "Build tests/examples")
set(DOCTEST_WITH_MAIN_IN_STATIC_LIB ON CACHE BOOL "Build a static lib for doctest::doctest_with_main")
set(DOCTEST_NO_INSTALL OFF CACHE BOOL "Skip the installation process")
set(DOCTEST_USE_STD_HEADERS OFF CACHE BOOL "Use std headers")

FetchContent_Declare(doctest
        GIT_REPOSITORY https://github.com/doctest/doctest.git
        GIT_TAG v2.4.11   # "main" for latest
        GIT_SHALLOW TRUE  # download specific revision only (git clone --depth 1)
        GIT_PROGRESS TRUE # show download progress in Ninja
        USES_TERMINAL_DOWNLOAD TRUE)
FetchContent_MakeAvailable(doctest)
```

Listing 29: ./matplotplusplus.txt

```
include(FetchContent)
FetchContent_Declare(gnuplot
        GIT_REPOSITORY https://git.code.sf.net/p/gnuplot/gnuplot-main gnuplot-gnuplot-main
        GIT_TAG origin/master)
FetchContent_Declare(matplotplusplus
        GIT_REPOSITORY https://github.com/alandefreitas/matplotplusplus
        GIT_TAG origin/master)
FetchContent_GetProperties(matplotplusplus)
if(NOT matplotplusplus_POPULATED)
    FetchContent_Populate(matplotplusplus)
    add_subdirectory(${matplotplusplus_SOURCE_DIR} ${matplotplusplus_BINARY_DIR} EXCLUDE_FROM_ALL)
endif()
```

Listing 30: ./README.md

```
# sP exam project
The project is compiled using
```