sP Exam Mini-project Assingment

Jonas Jerup Svare

June 4, 2023

This project has been developed on Windows 10, version 21H2, with the CLion compiler, version 2023.1.3. It does not require and specific tool or libraries to run.

Output figures can be found in Section 1.

Listing 1: CMakeLists.txt

```
cmake_minimum_required(VERSION 3.25)
project(Exam)

set(CMAKE_CXX_STANDARD 23)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fopenmp")

add_executable(Exam main.cpp lib/lib.cpp lib/lib.h lib/graph.cpp lib/graph.h)
```

Listing 2: lib.cpp

```
//
// Created by jonas on 31-05-2023.
//

#include <random>
#include <iostream>
#include <algorithm>
#include "lib.h"

//using namespace stochastic;
namespace stochastic {

    /** =================================
     *  AGENT CLASS
     *  =================================
     */

    /*
     * Req. 1
     * Operator overload (+)
     */
    std::vector<stochastic::Agent> stochastic::Agent::operator+(const stochastic::Agent &rhs) const {
        std::vector<stochastic::Agent> agents;
        agents.push_back(*this);
        agents.push_back(rhs);
        return agents;
    }

    /*
     * Req. 1
     * Operator overload (>>=)
     */
    stochastic::Reaction operator>>=(const Agent &lhs, const Agent &rhs) {
```

```cpp
        auto reaction = stochastic::Reaction();
        std::string out;

        out += lhs.name;
        reaction.agentVec.push_back(lhs);
        reaction.actionVec.emplace_back("increment");

        out += " -> ";

        out += rhs.name;
        reaction.agentVec.push_back(rhs);
        reaction.actionVec.emplace_back("increment");

        reaction.leftHandSide = std::vector<Agent>{lhs};
        reaction.rightHandSide = std::vector<Agent>{rhs};
        reaction.out = out;
        return reaction;
    }

    /*
     * Req. 1
     * Operator overload (>>=)
     */
    stochastic::Reaction operator>>=(const Agent &lhs, const std::vector<Agent> &rhs) {
        auto reaction = stochastic::Reaction();
        std::string out;

        out += lhs.name;
        reaction.agentVec.push_back(lhs);
        reaction.actionVec.emplace_back("increment");

        out += " -> ";

        for (int i = 0; i < rhs.size(); ++i) {
            if (i > 0) {
                out += " + ";
            }
            out += rhs[i].name;
            reaction.agentVec.push_back(rhs[i]);
            reaction.actionVec.emplace_back("increment");
        }

        reaction.leftHandSide = std::vector<Agent>{lhs};
        reaction.rightHandSide = rhs;
        reaction.out = out;
        return reaction;
    }

    /*
     * Req. 1
     * Operator overload (>>=)
     */
    stochastic::Reaction operator>>=(const std::vector<Agent> &lhs, const Agent &rhs) {
        auto reaction = stochastic::Reaction();
        std::string out;
        for (int i = 0; i < lhs.size(); ++i) {
            if (i > 0) {
                out += " + ";
            }
            out += lhs[i].name;
            reaction.agentVec.push_back(lhs[i]);
            reaction.actionVec.emplace_back("decrement");
        }

        out += " -> ";

        out += rhs.name;
        reaction.agentVec.push_back(rhs);
        reaction.actionVec.emplace_back("increment");

        reaction.leftHandSide = lhs;
        reaction.rightHandSide = std::vector<Agent>{rhs};
        reaction.out = out;
        return reaction;
    }

    /*
     * Req. 1
     * Operator overload (>>=)
     */
    stochastic::Reaction operator>>=(const std::vector<Agent> &lhs, const std::vector<Agent> &rhs) {
        auto reaction = stochastic::Reaction();
        std::string out;
        for (int i = 0; i < lhs.size(); ++i) {
            if (i > 0) {
                out += " + ";
            }
            out += lhs[i].name;
            reaction.agentVec.push_back(lhs[i]);
            reaction.actionVec.emplace_back("decrement");
        }

        out += " -> ";

        for (int i = 0; i < rhs.size(); ++i) {
            if (i > 0) {
                out += " + ";
            }
            out += rhs[i].name;
            reaction.agentVec.push_back(rhs[i]);
            reaction.actionVec.emplace_back("increment");
        }

        reaction.leftHandSide = lhs;
        reaction.rightHandSide = rhs;
        reaction.out = out;
        return reaction;
    }

    /*
     * Req. 1
     * Operator overload (>>=)
     * Used for reactants that decay into environment
     */
    stochastic::Reaction operator>>=(const Agent &lhs, const std::string &env) {
        auto reaction = stochastic::Reaction();
        std::string out;

        out += lhs.name;
        reaction.agentVec.push_back(lhs);
        reaction.actionVec.emplace_back("increment");

        out += " -> " + env;

        reaction.leftHandSide = std::vector<Agent>{lhs};
        reaction.out = out;
        return reaction;
    }

    /** =================================
     *  MONITOR CLASS
     *  =================================
     */

    void stochastic::Monitor::insert (double &time, SymbolTable<Agent> &table) {
        auto it = monitorMap.find(time);

        if (it != monitorMap.end()) {
            auto tableVec = it -> second;
            tableVec.push_back(table);
            monitorMap[time] = tableVec;
        }
        else {
            monitorMap[time] = std::vector<SymbolTable<Agent>> {table};
        }
    }

    void stochastic::Monitor::fileStream(const std::string& filePath) {
        std::ofstream outFile (filePath);

        if (outFile.is_open()) {
            outFile << "time,agentname,agentamount" << std::endl;

            for (auto &mapping : monitorMap) {
                for (auto &table : mapping.second) {
                    for (auto &agent : table.fetchTable()) {
                        outFile << std::to_string(mapping.first) + "," + agent.first + "," + std::to_string(agent.second.amount) << std::endl;
                    }
                }
            }
            outFile.close();
        }
    }

    int stochastic::Monitor::estimatePeak() {
        auto peak = 0;
        for (auto &mapping : monitorMap) {
            for (auto &table: mapping.second) {
                auto H = table.get("H");
                if (H.amount > peak) {
                    peak = H.amount;
                }
            }
        }

        std::cout << "Estimated peak of hospitalized agents: " + std::to_string(peak) << std::endl;
        return peak;
    }

    void stochastic::Monitor::multiplyH() {
        for (auto &mapping : monitorMap) {
            for (auto &table: mapping.second) {
                auto H = table.get("H");
                H.amount *= 1000;
                table.insert("H", H);
            }
        }
    }

    /** =================================
     *  ALGORITHM CLASS
     *  =================================
     */

    double stochastic::Algorithm::computeDelay(stochastic::Reaction &r,   ⤦
↪stochastic::SymbolTable<Agent> &table) {
        std::random_device rd;
        std::mt19937 gen(rd());

        double lambdaK = 1.0;
        for (auto &agent: r.leftHandSide) {
            lambdaK *= table.get(agent.name).amount;
        }
        lambdaK *= r.lambdaRate;

        std::exponential_distribution<double> expDist(lambdaK);
        return expDist(gen);
    }

    bool stochastic::Algorithm::amountChecker(const stochastic::Reaction &reaction,   ⤦
↪stochastic::SymbolTable<Agent> &table) {
        for (auto &agent: reaction.leftHandSide) {
            if (!(agent.amount <= table.get(agent.name).amount)) {
                return false;
            }
        }
        return true;
    }

    /**
     * Req. 4
     * This is the stochastic simulation
     * @param reactionVec Used as the set of reactions to compute
     * @param endTime Defines the end time
     * @param agentVec Used to create our symbol table for
     * @param filePath The path to where we want to save the state monitor
     * @return
     */
    stochastic::Monitor stochastic::Algorithm::simulation(
            std::vector<stochastic::Reaction> &reactionVec,
            double endTime,
            std::vector<stochastic::Agent> &agentVec,
            const std::string& filePath)
    {
        double t = 0.0;
        Monitor monitor;
        stochastic::SymbolTable<stochastic::Agent> table =   ⤦
↪stochastic::SymbolTable<stochastic::Agent>::generateSymbolTable(agentVec);

        while (t <= endTime) {
            stochastic::Reaction minDelayRec = stochastic::Reaction();
            for (auto &reaction: reactionVec) {
```

```cpp
                reaction.delay = stochastic::Algorithm::computeDelay(reaction, table);

                if (reaction.delay < minDelayRec.delay) {
                    minDelayRec = reaction;
                }
            }

            t += minDelayRec.delay;

            if (all_of(minDelayRec.leftHandSide.begin(), minDelayRec.leftHandSide.end(),
                       [&table](const auto &agent) { return table.get(agent.name).amount > 0; })) {
                for (stochastic::Agent &r: minDelayRec.leftHandSide) {
                    auto a = table.get(r.name);
                    a.amount -= 1;
                    table.insert(r.name, a);
                }
                for (stochastic::Agent &p: minDelayRec.rightHandSide) {
                    auto a = table.get(p.name);
                    a.amount += 1;
                    table.insert(p.name, a);
                }
            }
            //TODO: print/save/monitor state
            monitor.insert(t, table);
        }

        monitor.fileStream(filePath);
        return monitor;
    }

    /** =================================
     *  VISUALIZER CLASS
     *  =================================
     */

    /**
     * Req. 2
     * This function pretty prints the reaction network
     * @param reactionVec
     */
    void stochastic::Visualizer::prettyPrintReactions(const std::vector<Reaction> &reactionVec) {
        std::cout << "=======================================" << std::endl;
        for (auto &reaction: reactionVec) {
            std::cout << "Reaction: " + reaction.out + " || Rate: " + std::to_string(reaction.lambdaRate) << std::endl;
        }
        std::cout << "=======================================" << std::endl;
    }

    /**
     * Req. 2
     * This function generates a network graph of any given set of reactions
     * @param reactionVec
     * @param filePath
     */
    void stochastic::Visualizer::generateNetworkGraph(const std::vector<Reaction> &reactionVec, const std::string &filePath) {
        std::ofstream outfile(filePath);

        std::vector<std::string> nameVec;

        std::string numNode = "N";

        outfile << "digraph {" << std::endl;

        for (auto &reaction: reactionVec) {
            for (auto &lhs: reaction.leftHandSide) {
                if (!(std::find(nameVec.begin(), nameVec.end(), lhs.name) != nameVec.end())) {
                    outfile << lhs.name + " [shape=box];" << std::endl;
                    nameVec.push_back(lhs.name);
                }
                outfile << lhs.name + " -> " + numNode << std::endl;
            }
            for (auto &rhs: reaction.rightHandSide) {
                if (!(std::find(nameVec.begin(), nameVec.end(), rhs.name) != nameVec.end())) {
                    outfile << rhs.name + " [shape=box];" << std::endl;
                    nameVec.push_back(rhs.name);
                }
                outfile << numNode + " -> " + rhs.name << std::endl;
            }
            outfile << numNode + " [label=\"" + std::to_string(reaction.lambdaRate) + "\"];";
            numNode += "1";
        }

        outfile << "}" << std::endl;
        outfile.close();
    }
}
```

Listing 3: lib.h

```cpp
//
// Created by jonas on 31-05-2023.
//

#ifndef EXAM_LIB_H
#define EXAM_LIB_H

#include <string>
#include <vector>
#include <map>
#include <cfloat>
#include <fstream>

namespace stochastic {
    class Reaction;

    class Agent {
    public:
        std::string name;
        int amount = 0;

        Agent() = default;

        Agent(const std::string& name, int amount) {
            this->name = name;
            this->amount = amount;
        }

        //Req. 1
        std::vector<Agent> operator+(const Agent &other) const;
        friend stochastic::Reaction operator >>= (const Agent &lhs, const Agent &rhs);
        friend stochastic::Reaction operator >>= (const std::vector<Agent> &lhs, const Agent &rhs);
        friend stochastic::Reaction operator >>= (const Agent &lhs, const std::vector<Agent> &rhs);
        friend stochastic::Reaction operator >>= (const std::vector<Agent> &lhs, const std::vector<Agent> &rhs);
        friend stochastic::Reaction operator>>=(const Agent &lhs,  const std::string &env);

    };

    class Reaction {
    public:
        std::vector<stochastic::Agent> agentVec;
        std::vector<std::string> actionVec;

        std::vector<Agent> leftHandSide;
        std::vector<Agent> rightHandSide;

        double lambdaRate{};
        double delay = DBL_MAX;

        std::string out;

        Reaction() = default;

        Reaction(Reaction const &reaction, double lambdaRate) {
            *this = reaction;
            this->lambdaRate = lambdaRate;
        }
    };

    /**
     * Req. 3
     * This class allows us to store an instance of the reaction network, using Agent objects, in a symbol table
     * @tparam T
     */
    template <typename T>
    class SymbolTable {
        std::map<std::string, T> table;

    public:
        void insert (const std::string &key, const T &value) {
            table[key] = value;
        }

        void update (const std::string &key, const T &value) {
            auto previousValue = get(key);
            auto newValue = previousValue + value;
            table[key] = newValue;
        }

        T& get (const std::string &key) {
            auto it = table.find(key);
            if (it != table.end()) {
                return it->second;
            }
            else {
                throw std::invalid_argument("Does not exist");
            }
        }

        bool contains (const std::string &key) const {
```

```cpp
            return table.count(key) != 0;
        }

        void remove (const std::string &key) {
            table.erase(key);
        }

        void PrintAll () {
            for (auto val : table) {
                std::cout << "Name: " << val.first << " Amount: " << val.second.amount << std::endl;
            }
            std::cout << "--------------------------" << std::endl;
        }

        std::map<std::string, T> fetchTable() {
            return table;
        }

        static SymbolTable<T> generateSymbolTable(const std::vector<T>& inputVec) {
            auto symbolTable = SymbolTable<T>();
            for (auto &input : inputVec) {
                symbolTable.insert(input.name, input);
            }
            return symbolTable;
        }

    };

    /**
     * Req. 7
     * This class lets us instantiate a state monitor consisting of several symbol tables.
     * This allows us to store our reactions over a timespan.
     */
     class Monitor {
         std::map<double, std::vector<SymbolTable<Agent>>> monitorMap;
     public:
         Monitor() = default;

         void insert (double &time, SymbolTable<Agent> &table);
         void fileStream(const std::string& fileName);
         int estimatePeak();
         void multiplyH();
     };

    class Algorithm {
    public:
        static double computeDelay(stochastic::Reaction &r, stochastic::SymbolTable<Agent>& table);
        static bool amountChecker(const stochastic::Reaction& reaction, stochastic::SymbolTable<Agent>& table);
        static stochastic::Monitor simulation(std::vector<stochastic::Reaction> &reactionVec, double endTime, std::vector<stochastic::Agent> &agentVec, const std::string& filePath);
    };

    class Visualizer {
    public:
        //Req. 2
        static void prettyPrintReactions(const std::vector<Reaction> &reactionVec);
        static void generateNetworkGraph(const std::vector<Reaction>& reactionVec, const std::string &filePath);
    };

```

```
}

#endif //EXAM_LIB_H
```

1 Outputs

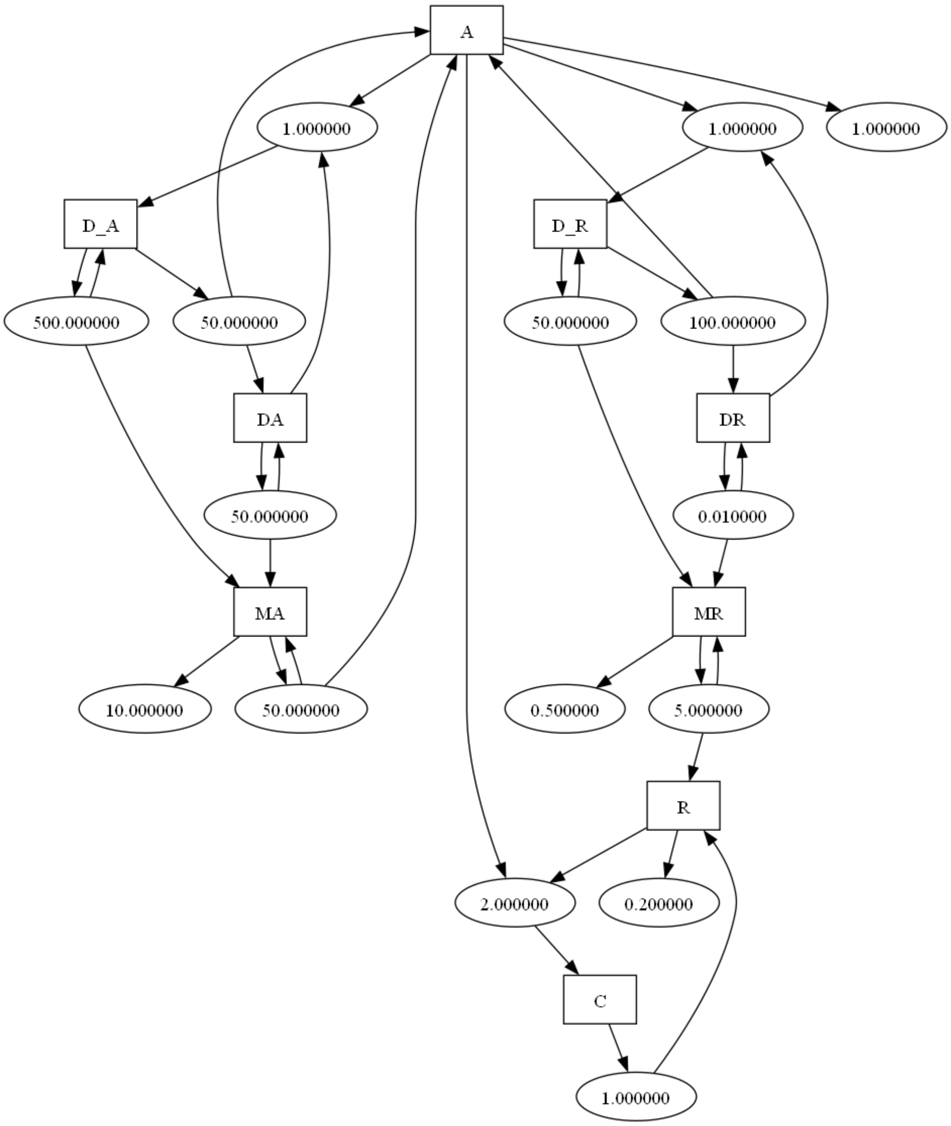

Figure 1: Network graph of the circadian oscillator example

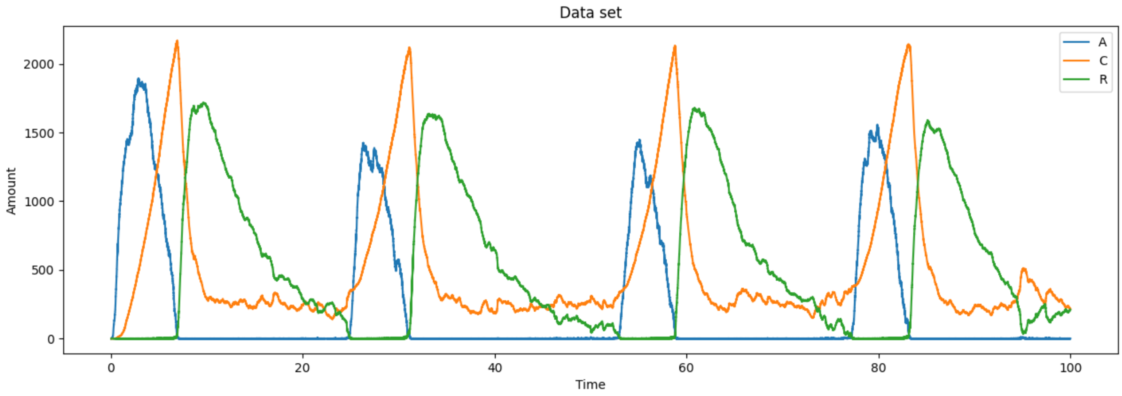

Figure 2: Output plot of the circadian oscillation example

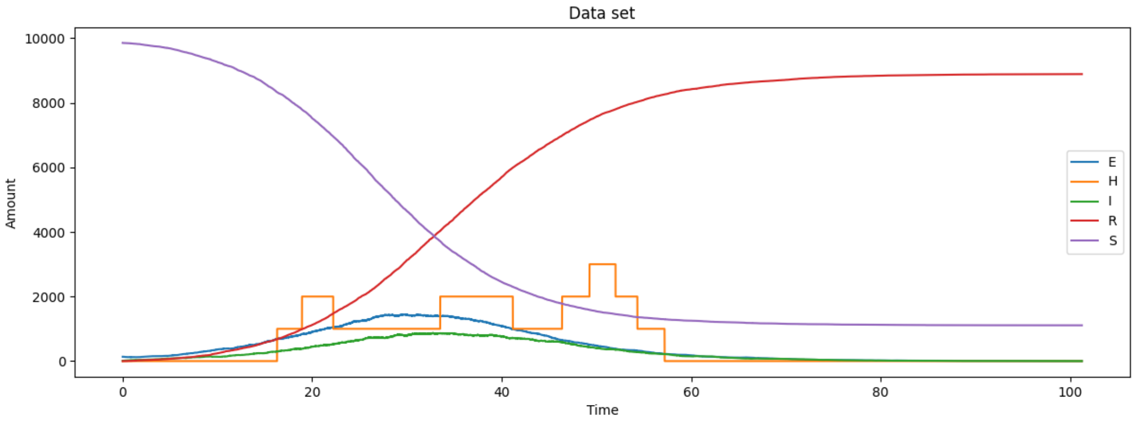

Figure 3: Output of the covid 19 exmaple with N = 10.000

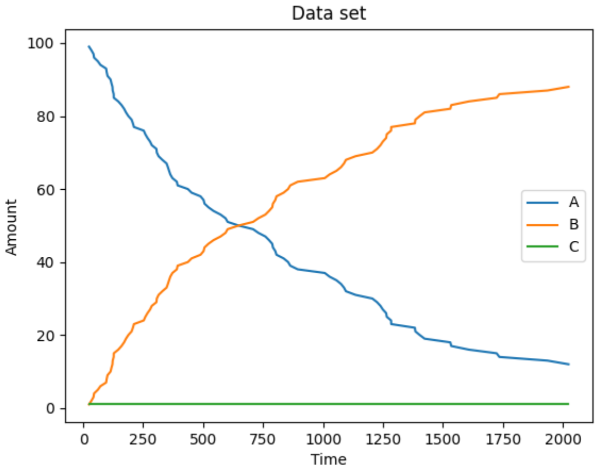

Figure 4: Output of figure 1, where A=100, B=0, C=1

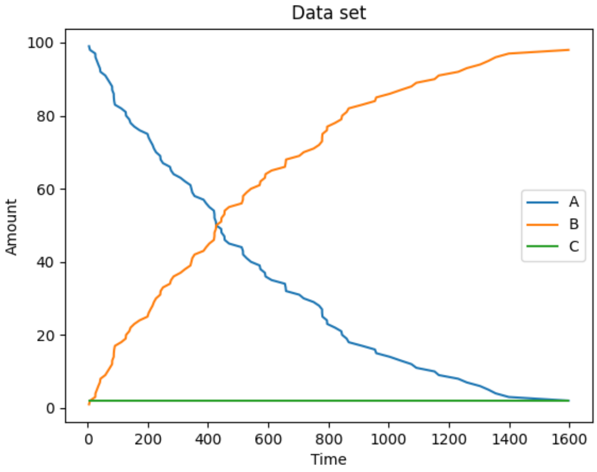

Figure 5: Output of figure 1, where A=100, B=0, C=2

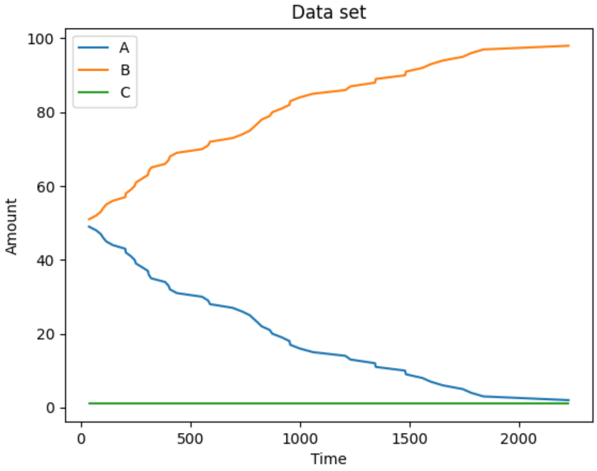

Figure 6: Output of figure 1, where A=50, B=50, C=1