MSc Project proposal: *Automata simulator*

Student: Dimitar Yanev

Supervisor: prof. Michael Zakharyaschev

Abstract

Automata theory is an important branch of computer science. Established back in the 20th century, automata theory deals with the logical computation of a machine called **automata.**

Automata is a finite state machine model with only one purpose, performing computation based on sequence of symbols by moving through series of states. At each stage of the computation a transition function determines the next step based on the architecture of the machine. The out come of the process e.g., whether it was successful or not, depends on the type of last state, if the state is final than the input is accepted and the computation is successful and if not the input is rejected [1].

The first description of finite automata was introduced by two neurobiologists Warren McCulloch and Walter Pitts in 1943, which purpose was to describe the behavior of the human brain through a model of neurons and synapses. The attempt contributed significantly in the domain of neuro network theory, theory of automata, the theory of computation and cybernetics [2].

The primary aim of this project is to develop and implement ***Visual Centric Automata Simulator Accepting Diagrammatic Input type*** simulating automata application, written in Java, which covers the three main type of finite state machine e.g. Pushdown automata (PDA), Non-deterministic finite automata (NDFA) and Deterministic finite automata (DFA). Also to provide user with a tool for visualizing and interacting with theoretical concepts.

Main objective of the project is to provide a simulation software, which allows the user to construct and simulate automata models.

Introduction

Due to its immense importance in the domain of computation, automata theory has become an irreplaceable part of every computer science curriculum. The problem rises in the traditional way of teaching e.g. solving problems are conducted only using pen and paper, approach which frustrates the majority of students due to lack of visual model to compare and test their solutions, which leads to bad results accompanied with poor comprehension and understanding of the importance of formal languages and automata.

Audience

The software product of this project will be suitable for students who study Computer science (and indeed, anyone else) interested in exploring the vast field of computing or more precisely the domain of automata theory and formal languages.

1. Literature review

1.1 Background research

Modelling & Simulation:

Modelling: is a process of constructing a simplified model similar to a real system or object, containing all related properties, which aids analyst to predict the effect of changes to the system.

Simulation: is a process of utilizing a model in terms of time or space, which helps to analyze the way how a particular system performs and behaves [11].

Components of a model:

***Entity* –** A representation of an object or a system. There are two types:

- **Dynamic –** It moves through the system.
- **Static –** Services other entities.

***Attribute* –** Local variables, which define the characteristics or properties of an entity.

***Activity* –** Any process causing changes in a system is called activity.

***State* –** A collection of variables called state variables, which define happening an entity or a system at any given point of time.

***Event* –** An occurrence which changes the state of a system.

There are two main types of models ***Mathematical*** and ***Physical***

1.1.1 Physical model

Physical model is a smaller or bigger physical copy of an object. The object may be small (an atom) or big (solar system). This type of model allows visual representation of an entity.

1.1.2 Mathematical model

In order to represent a model of this type, as the name suggests, mathematical equations and symbolic notations are used.

There are two type of systems Discrete and Continues - event systems.

Discrete-event system is a discrete – state, event – driven system. Typical feature of discrete - event systems is the state variables change only at discrete points of time when an event occurs.

Continues-event system. In this type of system the state variables are controlled by continuous functions and change continuously, without any delay, with respect to time.

Mathematical model itself is classified into five different types.

1.1.1.1 Static model.

Representation of a system at a particular point in time, in other words time does not have any effect. Classical example is *Monte Carlo Simulation.*

1.1.1.2 Dynamic model.

It is the opposite of the static model. Representation of a system as it evolves over time i.e. time plays critical role.

1.1.1.3 Deterministic model.

Typical for this type is that, they do not contain any random variables or any degree of randomness. In such a model a given input will always produce the same output.

1.1.1.4 Stochastic model.

Stochastic models embraces the randomness. This type incorporates one or more probabilistic elements into the model. Every time when the model is executed the result will be different even with the initial conditions.

1.1.1.5 Hybrid model

Hybrid model combines two or more from the types described above.

[14]

1.2 Existing technology

Due to the immense importance of automata theory in the field of computer science, various automata simulators have been developed and used for educational and academic purposes around the world for the last fifty years. Although, the big number of already existing simulating tools for simulating automata exists, the scientific community is always open for new and better ones, which will continue to contribute and improve to the teaching process.

Automata simulators have been classified into two major groups based on their design paradigm. ***Language based automata simulators*** and ***Visualization centric Automata simulators***.

1.2.1 Language based automata simulators.

This approach to simulate automata is the older one of the two basic approaches. In order to define an automata. First the user has to write it down as a program, using symbolic language

and second to process the program. There are two approaches which are used to process the program.

In the first approach the program is firstly compiled into intermediate language and checked for errors. Secondly an interpreter is used to create the simulation process upon input string of symbols.

In the second approach the program is directly loaded into the interpreter, which is responsible for conducting lexical, sintaxical and error free analysis.

The language based automata simulators are divided into four subgroups.

1.2.1.1 Notational language based automata simulators.

This type of simulators the automata is defined by using short symbols. Notational languages are easy to learn and used to define simple type of automata. Drawbacks are, flow control and constructing of data abstraction are not supposed.

Perhaps the first ever build automata theory simulating software tool based on Notational language was done by Coffin [3]. Its purpose was to simulate a Turing machine which was represented in the form of quintuples, each denoting transition. Another program called “The builder” transforms the quintuples into machine language. Then a third program called “The driver” simulates the performance of the Turing machine.

```
DFA//Type
DIV5//Title
0 1//Input alphabet
qi q0 q1 q2 q3 q4//States
qi//Initial state
q0//Final state
qi 0 q0//Transitions
qi 1 q1
q0 0 q0
q0 1 q1
q1 0 q2
q1 1 q3
q2 0 q4
q2 1 q0
q3 0 q1
q3 1 q2
q4 0 q3
q4 1 q4
end
```

Figure 1: Deterministic finite automata simulator accepting all binary numbers which equivalents are divided by 5.

Another three simple simulators are developed by Head [4]. The set contains a Finite State Machine Simulator, Non-deterministic Pushdown Automata simulator and Turing machine. Four different versions of the simulators are available i.e. graphical, short text only, full text only and course grader’s quick to use.

1.2.1.2 Assembly -like base automata simulators.

This type of simulators contains simple set of instructions. Return instructions, conditionals and unconditional jumps and subroutine calls govern the execution of the program.

Even though, Assembly - like based simulators are more efficient than Notational language based simulators in building bigger and more complex automata models, they also share the same drawback i.e. inability to construct data abstraction.

A Turing machine was developed by Curtis [5] in order to be used as a teaching and researching tool at Wesleyan University. The simulator contains two pass loaders, the first one sets up the symbol table and the second one loads the instructions. The simulator was executed on IBM 1620 machine.

1.2.1.3 Procedural language based automata simulators.

By nature Procedural languages are much more powerful than assembly languages. Providing a various high level features for flow control and data abstraction, Procedural languages can be used for defining a very large and complex automata.

The power of Procedural languages to build an automata simulators was demonstrated by Knuth and Bigelow [6]. They constructed a simulator which simulates a Pushdown Automata and also demonstrated how easily is to adapt the simulator into different forms of automata.

1.2.1.4 Descriptive language based automata simulators.

In contrast with notational, assembly – like and procedural languages which need some efforts to learn, descriptive languages, defined by, Chakraborty [10], have the potential to allow the user to build a model written in a formal textbook – like way, without the need to spend hours learning the features of the language.

```
FA=({qi,q0,q1,q2,q3,q4},
{0,1},D,qi,{q0});
D(qi,0)=q0,
D(qi,1)=q1,
D(q0,0)=q0,
D(q0,1)=q1,
D(q1,0)=q2,
D(q1,1)=q3,
D(q2,0)=q4,
D(q2,1)=q0,
D(q3,0)=q1,
D(q3,1)=q2,
D(q4,0)=q3,
D(q4,1)=q4;
```

Figure 2: Definition of deterministic finite automata using descriptive language.

A fast single-pass compiler is used to compile automata models written in descriptive language. The compiler itself consist of four parts lexical, syntax, sematic analyzer and code generator. After the compiling process a suitably designed interpreter simulates the compiled model.

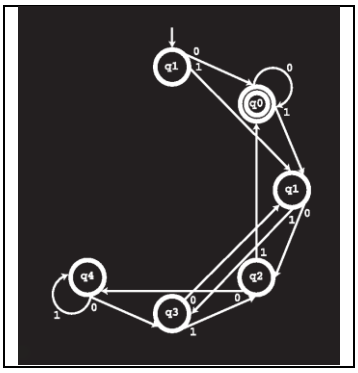


Figure 3: Transition diagram of deterministic finite automata.

Additional tools to display the transition diagram of the compiled automata, converting Nondeterministic Finite Automata into Deterministic Finite Automata and Deterministic Finite Automata into Turing machine are also available.

1.2.2 Visualization centric automata simulators.

Distinctive feature of the Visualization simulators is the ability to demonstrate a working model usually accompanied with high quality graphics and animations.

Automata specifications are accepted as an input in predefined structured or diagrammatic forms.

Simple tools like adjusting simulating speed and converting from one type to another are also available.

According to the nature of the input the Visualization centric automata simulators are clarified into two groups.

1.2.2.1 Visualization centric automata simulator accepting structured input.

The user is required to fill in a form providing all the details of a desired automata model. This type of simulators are famous with their easy to use feature due to quite a few of them have been developed over the years.

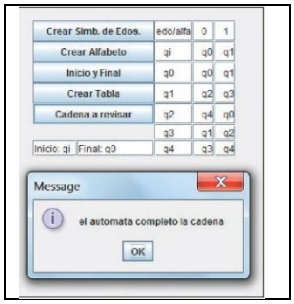


Figure 4: Simulation of deterministic finite automata using Automata en Java.

Typical simulator that accepts structured input is Automata en Java, a simple tool developed by Dominguez [7] to simulate a DFA.

1.2.2.2 Visualization centric automata simulator accepting diagrammatic input.

One of the most successful and popular type of simulators ever developed. Typically user is provided with a canvas where states and transactions are added and positioned by only clicking and dragging the mouse giving the user the experience of drawing automata on peace of paper. Due to its features Visualization centric automata simulators accepting diagrammatic input type has become the most favorite tool of simulating automata among students and teachers. As a result several of simulators have been the developed.

Among all the most famous and sophisticated tool is Java Formal Language Automata Package (JFLAP).

Developed by Roger and co – researchers [8 – 9] its distinctive and unique features made it irreplaceable part in nowadays teaching process. It is the most widely used for simulating automata.

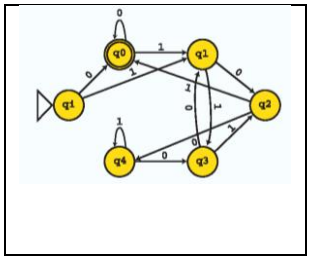


Figure 5: Deterministic finite automata using Java Formal Languages and Automata Packages (JFLAP).

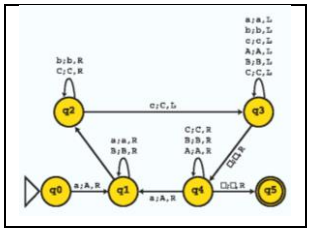


Figure 6: Turing machine using Java Formal Languages and Automata Packages (JFLAP).

JFLAP introduces to the user rich variety of options and friendly graphical interface, also it is the best documented tool for simulating automata. Here are some of the most important features that make this simulator so unique.

User is able to visually design a model e.g. draw, edit and simulate its work. The variety of models that can be design and simulate on JFLAP are: NAF, DFA, PDA, Turing machine, Mealy machine and Moore machine.

Another feature of the tool is, converting from NDFA to DFA to minimal DFA.

Simulation can either be run in step – by – state mode, fast run mode or multiple run mode.

2. Project details

2.1 Goals

Main objectives of this project are:

- Create easy to use and friendly interface.

User will have the ability to:

- Choose what type of automata to create and simulate (NDAF, DFA and PDA).
- Choose what type alphabetic symbols to use by his/her own desire.
- Choose the number of symbols in the alphabet.
- Establish states and set their name and type i.e. initial or final.
- Create and add transitions between states by simply dragging from one state to another.
- Eddie existing transitions and states.
- Delete states and transitions.
- Create numerous input strings of symbols and test whether the string is accepted or rejected by the constructed automata model.
- Save his/her own models of automata to a XML file and load them for later use.
- Access to already constructed sample models in the form of XML file which can be loaded and simulated.

During simulation the user will be able to choose:

- Simulation speed.
- Whether to run the simulation automatically manually (step by step).
- Whether to observe the

Depending on the users choose what type of automata to construct and simulate:

Pushdown Automata

User will be able to observe the model's behavior during simulation in the form of animation.

- On one side of the working window there will be a stack explicitly displaying what has been pushed and popped.

- On the bottom of the window there will be an array showing the input string and a pointer pointing to a particular symbol at a particular point of time.

- In the middle of the window the actual model will be displayed.

Nondeterministic and Deterministic Finite Automata

The graphical representation of NDFA and DFA will be similar to PDA apart from displaying the stack.

3. Tools and programming language

3.1 Programming language:

Main programing language will be Java.

3.2 Tools:

- IntelliJ IDEA will be the integrated development environment.

- Java Virtual Machine (JVM).

- Junit 4 (for conducting Test Driven software development process)

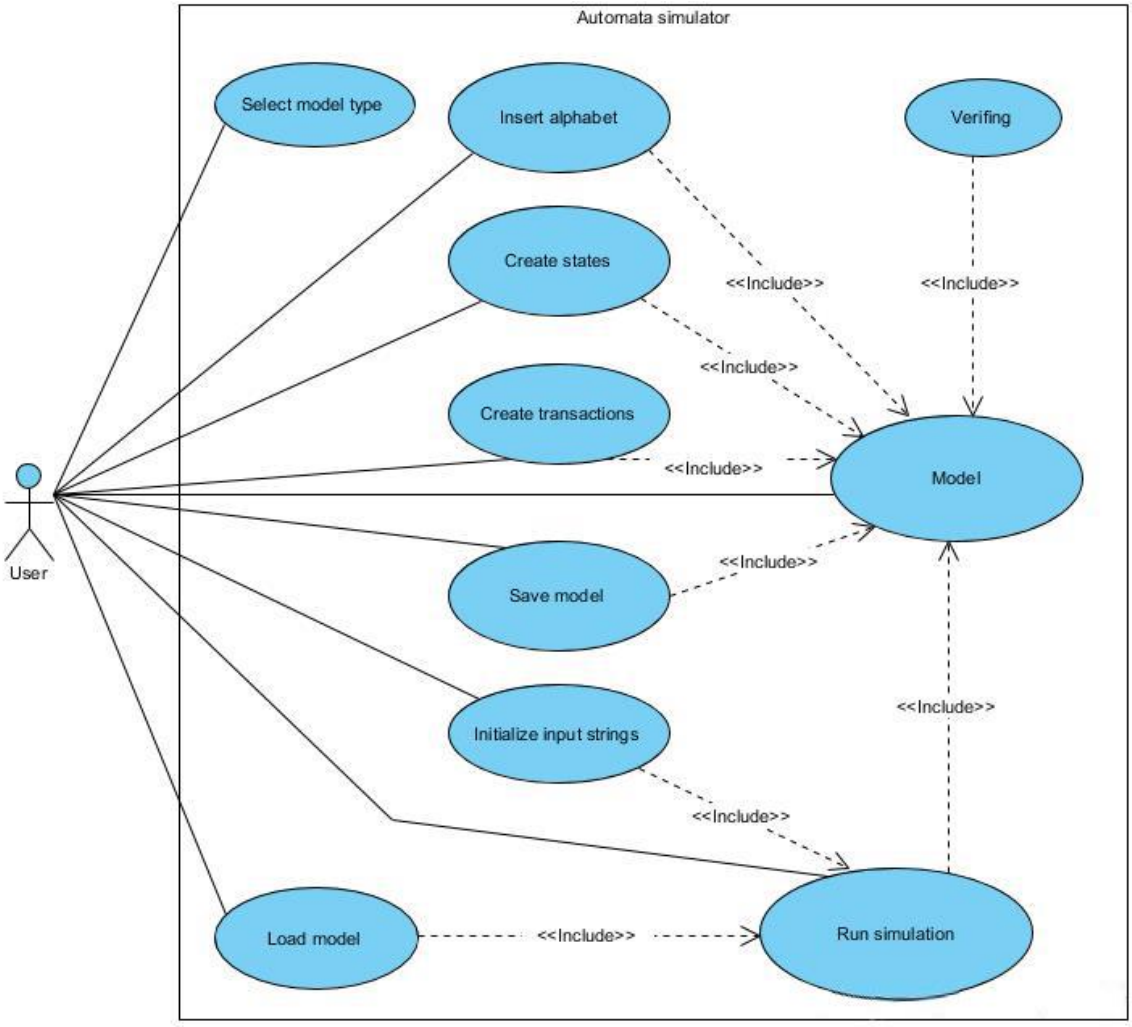
- Photoshop CC (for creating animations and visual representation of the model and its components)

- Window 10 operating system

- Visual Paradigm for UML Community Edition (for representing Use case, Behavioral, Class and object diagrams).

4. Software architecture

Use case diagram:



5. Methodology & work plan

5.1 Methodology

Chosen methodology: ***Agile methodology.***

Type of Agile Methodology: ***Kanban***

Kanban is a very simple methodology developed by Toyota to help increase the productivity in factories. At its core Kanban can be thought as a prioritized to – do list.

The main difference between Scrum and Kanban is that, Scrum is time based and Kanban is priority based methodology. When a developer is finished with a particular task and ready for the next one, she/he pulls it from the to – do list [12].

5.2 Work plan

Designing a simulation software involves four main phases [13].

- ***Designing a conceptual model.***
- ***Building a simulation engine***
- ***Verification***
- ***Validation***

Due to the nature of this simulator, which will not contain built in model (apart from already saved sample models), Instead of building an actual conceptual model, the main aims of this phase will be to construct software and tools with which the user will be able to build the actual model (interfaces and classes responsible for creating states, transitions, alphabet etc.) needed to construct one will be designed.

During the second phase the software responsible for running the simulation will be built. In this case there will be three distinct simulation engines each responsible for simulating a different automata model.

Verifying the product will be done, to ensure that the software is being built according to the design patterns and specified requirements.

Validating the final product. This is the most important phase, one of the real problems is to validate the model. Conducting the validating process will include the following technics:

- Comparing various results to other results obtained from already existing software (JFLAP) using the same input.
- Discussing the model with system experts while in designing phase.
- Approving the final result by system expert.

6. Time table

Week 1	Establishing interfaces and classes needed for constructing entities and attributes (States, alphabet, etc.).
Week 2	Creating interfaces and classes needed for constructing the actual model (Automata).
Week 3 – 4 – 5	Developing the simulation engines.
Week 6	Creating the visual representation of the model and its attributes (States, transitions, transition tables etc.) using Photoshop CC.
Week 7	Forging animations to represent the simulation process in Photoshop CC.
Week 8	Working on the graphical manifestation of the simulator using GUI library (Buttons, window frames etc.).
Week 9	Verifying the product including refactoring and debugging if needed.
Week 10	Validating the final software.

References

[1] https://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/automata-theory/basics.html

[2] Dominique Perrin, Universit´e de Marne-la-Vall´ee, *"Automata and formal languages"* July 15, 2003.

[3] Coffin, R. W., Goheen, H. E. and Stahl, W. R. 1963. Simulation of a Turing machine on a digital computer. Proceedings of the Fall Joint Computer Conference, pp. 35-43.

[4] Head, E. F. S. 1997. ASSIST: A Simple Simulator for State Transition. http://www.cs.binghamton.edu/~software/ASSIST.html.

[5] Curtis, M. W. 1965. A Turing machine simulator. Journal of the Association of Computing Machinery, 12(1): 1-13.

[6] Knuth, D. E. and Bigelow, R. H. 1967. Programming languages for automata. Journal of the Association of Computing Machinery, 14(4): 615-635.

[7] Dominguez, A. E. O. 2009. Automata. http://torturo.com/wp content/uploads/Automata.jar.

*[*8] Procopiuc, M., Procopiuc, O. and Rodger, S. H. 1996. Visualization and interaction in the computer science formal languages course with JFLAP. Proceedings of the Frontiers in education Conference, pp. 121-125.

[9] Rodger, S. H., Wiebe, E., Lee, K. M., Morgan, C., Omar, K. and Su, J. 2009. Increasing engagement in automata theory with JFLAP. inroads – ACM SIGCSE Bulletin, 41(1): 403-407.

[10] Chakraborty, P. 2007. A language for easy and efficient modelling of Turing machines. Progress in Natural Science, 17(7): 867-871.

[11] Ernest H. Page, Jr., September 1994 Blacksburg, Virginia. Simulation modelling methodology: Principles and etiology of decision support.

[12] https://www.qasymphony.com/blog/agile-methodology-guide-agile-testing/

[13] James J. Nutaro 2011. Building Software for Simulation. Theory and algorithms, with applications in C++, pp. 2.

[14] Kai Velten. Mathematical Modeling and Simulation.