Team Project - Team C5

Test Report

24.03.2016

Thomas Clarke, Mihnea Patentasu, Piotr Wilczyński, Danyil Ilchenko, Owen Pemberton, Botond Megyesfalvi

Table of Contents

Test Plan

Introduction

This test report documents the testing of our automata builder project. Its aim is to make sure the team has tested the functionality of the project and met all the functional requirements. It describes the approach and the criteria used. It contains detailed information about our functional testing as well as usability testing.

Test Items

The scope of the testing is based on the updates specification. It includes all the components that users can interact with.

- Inputting a regular expression
- Creating a non-deterministic finite automaton
- Determinising the automaton
- Viewing two automata at the same time
- Saving, loading and printing automata
- Checking if a word is accepted by the automaton

Approach

The project has three testing methods: unit testing, integration testing and usability testing. JUnit testing has been conducted in order to check for errors that may arise when new functionalities are added to the software. Integration testing was done in order to make sure all the components are compatible and everything is displayed correctly in the user interface. Usability testing was run in order to measure the usability and effectiveness experienced by users and obtain feedback from them.

Pass/Fail Criteria

Unit Tests

For a JUnit test to pass, all the test cases have to run successfully. The purpose of this is to check for errors after changes have been made in the code and make sure all the components are still working correctly.

Integration Testing

For an integration test to pass, the Unit Tests for each individual component used have to pass in order to show that they are correctly integrated and communicate properly.

Usability Testing

There is no pass/fail criteria for this type of testing. It is based on users giving feedback about their experience with the system as well as making suggestions about its possible improvements.

Rationale for the test cases

There was a number of test cases in the project that the team conducted in order to make sure that all the components are working in accordance with their intended functionality and that they are fully compatible. They include unit testing such as checking individual methods in the code as well as testing the functionality of the interface. They were carefully chosen so that they cover all the important aspects of the project and detect flaws and errors that have appeared in the system.

Functional and Usability Testing

In order to make sure the system is functional and usable for the intended users, test cases were composed which can be found in the appendices. Furthermore, the team prepared questionnaires which were given to users along with the application with the aim of getting direct feedback from them and suggestions for changes and improvements of our system.

Unit Testing

JUnit tests can be found in the SVN repository under final/src/automaton/testing. This was performed to ensure that every single method is reliable, ready to use and behaves as expected.

Test Name	Class Tested	Test Description	Test Scenario	Actual Results	Overall Test Result
testType()	ConBlock	Check the type of a regular expression	The test runs getType() and asserts if the block is an ELEM, CONCAT, DISJ or STAR block	TRUE	PASS
testConcents()	ConBlock	Check the nested ConBlocks	The test checks if the regular expression is parsed correctly	TRUE	PASS
testToString()	ConBlock	Check if the regular expression is parsed correctly	The test checks if the regular expression is parsed correctly	TRUE	PASS
testIsWellFormed()	RegexParser	Check if the regular expression is a valid regexp	The test checks if the expression is well bracketed and well formed	TRUE	PASS
testNotWellFormednness()	RegexParser	Check if the regular expression is an invalid regexp	The test asserts if the regular expression is invalid	FALSE	PASS
testFrameCount()	Automaton	Check the number of frames of an automaton (NFA/DFA)	The test checks the numbers of frames in an automaton	TRUE	PASS

testNumberOfStates()	Automaton	Check the number of the states in an automaton	The test checks the number of states in an automaton	TRUE	PASS
testNumberOfAcceptingStates()	Automaton	Check if the number of accepting states is correct	The test checks if the automaton is generated with the correct number of accepting states	TRUE	PASS
testWordChecking()	Animation	Check if the word is accepted or not	The test checks if the automaton accepts a certain word	TRUE	PASS

Usability Testing

In order to evaluate our product, once all the application functions have been implemented, we conducted usability testing to ensure that our target audience can easily work with this learning tool. We asked the lecturers teaching modules related to the automata theory for feedback. The suggestions that they made were:

- Highlighting edges
- Showing states in the closure during the determinisation
- Accepting strings such as “ |a”

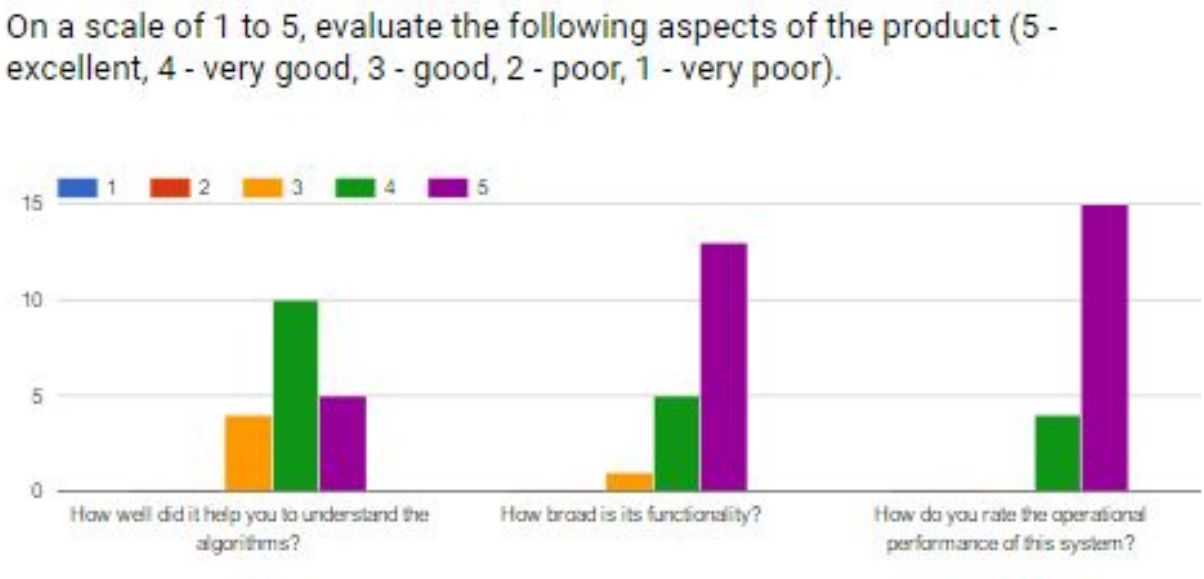
We have taken this into consideration thoroughly and adjusted our application to satisfy those requirements. This was the highest priority feedback as it was from people who will potentially use our product in the future for their teaching purposes.

The next step of our usability testing was to get direct feedback from our coursemates who are also currently studying a module related to automata theory and could be interested in using our system as a learning tool. We presented our application to them and we designed a questionnaire that they completed after using it. Once the results were collected, we reviewed their suggestions in detail and grouped them based on their relevance to our application’s functionality. The improvements that we judged most important and implemented were as follows:

- Adding a help page
- Adding a speed slider to increase/decrease the speed of the animation
- Adding a title for the current phase of the algorithm

- Showing what character the algorithm reached when it tests a word

The exact results of the questionnaire can be viewed on the diagram below and in the appendices. Those suggestions have been fully implemented in order to meet potential users' requirements.



The person we asked to fill in the questionnaire made a few further suggestions which unfortunately we have not had enough time to implement. They could be taken into consideration in the future development of the application. They were as follows:

- Expression ranges (e.g. a^3 meaning three a's)
- Zooming
- Randomising regular expressions
- Audio announcements

Generally the feedback was very positive and we could rest assured that our project would be suitable as a learning tool for students to learn more about the automata theory. As can be seen on the diagram in the appendix, in terms of the way the system deals with the algorithms most people gave the highest mark.

We also tested the colourblind mode on a colourblind person to make sure it is properly visible to colourblind people.

As that person said, without the colour blind mode it is extremely confusing to distinguish the meaning of the different colours (mainly green and red but it can also be hard to determine other colours). That is because they look the same when the changes are highlighted. Activating the colour blind mode makes green colours and red colours look separate, and the slight changes in other colours help make the

different colours on the screen stand out more which makes the whole program easier to use and understand.

Appendix A - Example Test Cases

Test Case ID	Input	Expected Output	Output	Pass?
TC_ME_01	Mouse click on the “Generate” button after a regular expression was inputted	The first building stage of the NFA generation is shown	First building stage of the NFA generation is shown	Yes
TC_ME_02	Mouse click on the generate button when there is an empty regular expression field	The NFA that accepts an empty string is shown	NFA that accepts an empty string	Yes
TC_ME_03	Mouse click on the “x” of the tab	The tab will be closed	Tab was closed	Yes
TC_ME_04	Mouse click on “File” tab from the toolbar	The File menu is shown	The File menu is shown	Yes
TC_ME_05	Mouse click on “New” from the “File” menu	A new tab is created	A new tab is created	Yes
TC_ME_06	Mouse click on “Save” from the “File” menu before generating an automaton	Nothing happens because there is no regular expression to save	Nothing happens	Yes
TC_ME_07	Mouse click on “Save” from the “File” menu after an automaton is generated	A new window where you can save and manage your regular expressions	A new window pops up where you can save and manage your regular expression	Yes

TC_ME_08	Mouse click on “Save” after no slot was selected	The window closes and the regular expression is saved in a new save slot	The windows closes and the regular expression is saved in a new save slot	Yes
TC_ME_09	Mouse click on “Save” when a previously save slot is selected	The window closes and the regular expression is overwritten on that slot	The window closes and the regular expression is overwritten on that slot	Yes
TC_ME_10	Mouse click on “Delete” when no save slot is chosen	No changes are made because no save slot is chosen	No changes are made because no save slot is chosen	Yes
TC_ME_11	Mouse click on “Delete” when a save slot is chosen	The previously saved slot is deleted	The previously saved slot is deleted	Yes
TC_ME_12	Mouse click on “Cancel”	The Save window is closed	The Save window is closed	Yes
TC_ME_13	Mouse click on “Open” from “File” menu	A new tab should be created and a new window called “Load a regular expression”	A new tab is created and a new window called “Load a regular expression” is created	Yes
TC_ME_14	Mouse click on “Load” and “Delete” when there are no saved slots	Nothing should happen because the table is empty	Nothing happens because there is not anything to load/delete	Yes
TC_ME_15	Mouse click on “Load” and “Delete” when there are no saved slots selected	Nothing should happen because there is no selected slot	Nothing happens because there is no selected slot	Yes
TC_ME_16	Mouse click on “Load” when a slot is selected	The newly created tab should load the selected regular expression and the “Load” window closes	The newly created tab loads the selected regular expression and the “Load” window closes	Yes

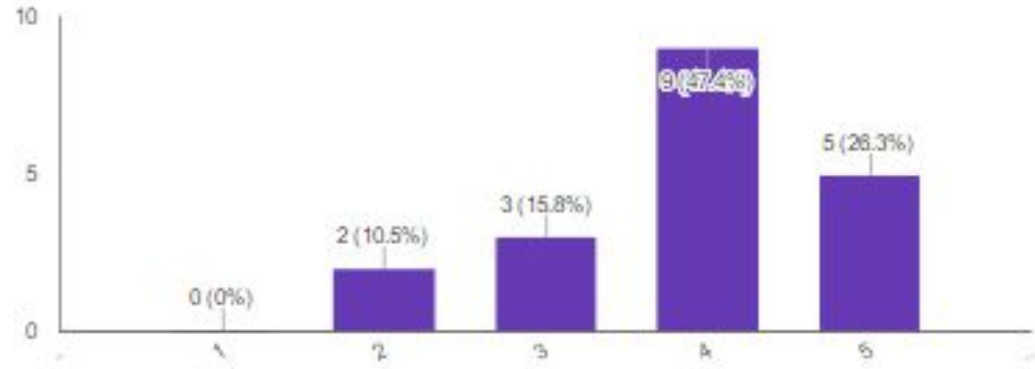
TC_ME_17	Mouse click on “Delete” when a slot is selected	The selected slot should be deleted	The selected slot is deleted	Yes
TC_ME_18	Mouse click on “Print” from “File” menu	A new window should be opened where you can print the automaton	A new window is opened where you can print the automaton	Yes
TC_ME_19	Mouse click on “Close” from “File” menu when there are open tabs	The currently selected tab should be closed	The currently selected tab is closed	Yes
TC_ME_20	Mouse click on “Close” from “File” menu when there are no open tabs	The application should close	The application closes	Yes
TC_ME_21	Mouse click on “Automata” tab from the toolbar	The submenu should appear	The submenu appears	Yes
TC_ME_22	Mouse click on “Test a Word” from “Automata”	It should open a new tab that requests a string	It opens a new tab that requests a string	Yes
TC_ME_23	Mouse click on “View Help” tab from the toolbar	The submenu should appear	The submenu appears	Yes
TC_ME_24	Mouse click on “Show Help” from “View Help”	A tab named “Help” should open up	The “Help” tab is opened	Yes
TC_ME_25	Mouse click on “Show Help” from “View Help” when there is already a “Help” tab open	It should redirects you to your open tab	It redirects you to your open tab	Yes

TC_ME_26	Mouse click on “Toggle Colour Blind” from “View Help”	It should switch the colour of the accepting state from green/red to red/green	It switches the colour of the accepting state from green/red to red/green	Yes
TC_ME_27	Mouse click on “About” from “View Help”	It should open a tab with information about the application	It opens a tab with information about the application	Yes
TC_ME_28	Mouse click on “►” button after a regular expression is written and an automaton is generated	The animation should go through each phase of the construction	The animation shows each stage of the automaton	Yes
TC_ME_29	Mouse click on “■” button when the animation plays	The animation should stop at the exact phase when the button was pressed	The animation stops at the exact phase when the button was pressed	Yes
TC_ME_30	Mouse click on "|◄◄" during the generation of an automaton	The animation should set back to the start of the generation	The animation is set back to the start of the generation	Yes
TC_ME_31	Mouse click on "►►|" during the generation of an automaton	The animation should skip to the final automaton	The animation skips to the final automaton	Yes
TC_ME_32	Mouse click on "►|" during the generation of an automaton	The animation should go to the next phase of generation	The animation goes to the next phase of generation	Yes
TC_ME_33	Mouse click on "|◄" during the generation of an automaton	The animation should go to the previous phase of generation	The animation goes to the previous phase of generation	Yes

TC_ME_34	Mouse click on "►►" during the generation of an automaton	The speed of the animation should increased	The speed of the animation is increased	Yes
TC_ME_35	Mouse click on "◄◄" during the generation of an automaton	The speed of the animation should decrease	The speed of the animation is decreased	Yes
TC_ME_36	Mouse click on “Next (Determinising)” button after the NFA was generated	It should take you to the initial stage of the DFA construction	It takes you to the initial stage of the DFA construction	Yes
TC_ME_37	Mouse click on “Previous (Generation)” button after the “Next (Determinising)” button was pressed	It should take you back to the initial stage of the NFA construction	It takes you to the initial stage of the NFA construction	Yes
TC_ME_38	Mouse click on “Run String” button when there is a String inputted	The DFA should be shown with the playback buttons that control the animation	The DFA is shown with the playback buttons that control the animation	Yes

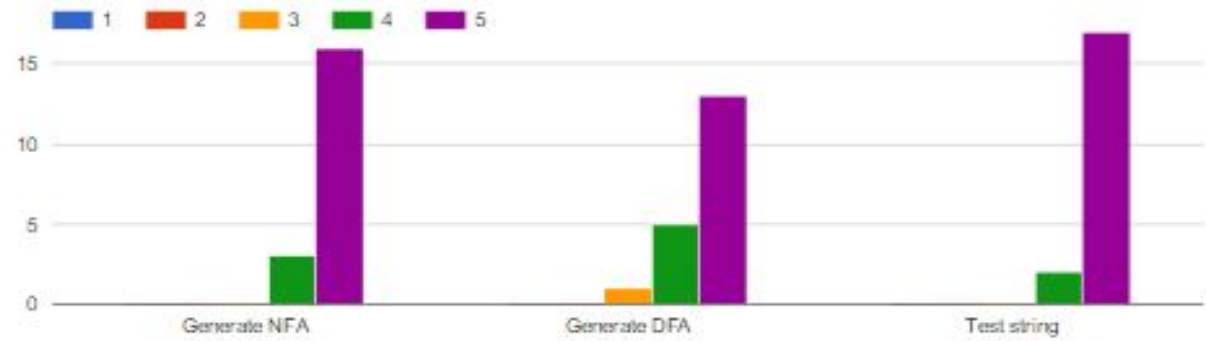
Appendix B - Questionnaire results

How well do you know automata? (19 responses)



On a scale of 1 to 5, evaluate the following aspects of the product (5 - excellent, 4 - very good, 3 - good, 2 - poor, 1 - very poor).

On a scale of 1 to 5, evaluate the way the system deals with the following algorithms.



Appendix C - Updated Specification

Team Project - Team C5

Software Requirements Specification

24.03.2016

Thomas Clarke, Mihnea Patentasu, Piotr Wilczyński, Danyil Ilchenko, Owen Pemberton, Botond Megyesfalvi

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) is to provide a detailed overview of our project based on algorithms related to the automata theory. It describes the aim of the project and its requirements. It is to explain the user interface and possible interactions with the user. This document is intended for review by relevant lecturers (as they will be using it for demonstration purposes) and for the production team as a basis for development.

1.2 Scope

The purpose of our system is to develop a learning tool to help Computer Science students learn about Finite State Automata and understand the Thomson's Construction and Subset Construction algorithm using an interactive visualisation. Users are able to input regular expressions and visualise the steps required for transforming them into deterministic finite automata. Users can also input a word and see how it is tested for being part of the language or not.

1.3 Definitions, Acronyms and Abbreviations

1.3.1 Definitions:

Deterministic Finite Automaton **(DFA)** — a finite state machine that accepts/rejects finite strings of symbols and only produces a unique computation (or run) of the automaton for each input string. 'Deterministic' refers to the uniqueness of the computation.

Nondeterministic Finite Automaton **(NFA)** — unlike a DFA, it is not deterministic, i.e., for some state and input symbol, the next state may be one of two or more possible states.

1.3.2 Acronyms:

DFA: Deterministic Finite Automaton

NFA: Nondeterministic Finite Automaton

GUI : Graphical User Interface

1.4 References

IEEE. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1998.

Ken Thompson (Jun 1968). "Programming Techniques: Regular expression search algorithm". *Communications of the ACM* **11** (6): 419–422.

Rabin M. O.; Scott D., (1959). "Finite automata and their decision problems". *IBM Journal of Research and Development* **3** (2): 114–125.

Xing, Guangming. “Minimized Thompson NFA”, [available at: http://people.wku.edu/guangming.xing/thompsonnfa.pdf]

1.5 Overview

The rest of this document will detail a general description of the overall product that shall be produced, along with more specific requirements about the system. This will also cover how the user will interact with the system. Furthermore, a risk analysis shall be supplied to help solve issues as they arise as various protocols shall already have been discussed and decided.

The SRS has been organised in the sequence listed above, so that it follows a logical order of describing the project, then explaining various parts of it and finishing with information to be used when actually building it.

2. General Description

2.1 Product Perspective

The system will be a standalone learning tool to be used by university students and lecturers in modules closely related to the automata theory. The system will be in the form of an application runnable on Linux, Mac and Windows desktops and laptops. The application will visualise the process of constructing automata from regular expressions, determinising nondeterministic automata and testing whether a word is accepted by the language.

2.2 Product Functions

With this application the user will be able to build a finite state automata. The user will type in a regular expression, the system will confirm whether it is valid or not, and the application will build the automaton. The user will be able to watch this whole process. The application will display the construction, going step by step with a certain speed that can be changed at any time, or even completely stopped by the user. The construction will pause at relevant stages of the construction (after generating the nondeterministic automata and after determinising). "Next" and "Previous" buttons will also be included, which will allow the user to go through every small step of the construction manually.

At any point during this construction process a saving and a printing option will be available, which will save as an image file or print the current automaton, with useful text added, which describes the current stage of the constructed automaton, and also the regular expression that it represents. The user will have the option to select the destination folder where to save the image, or view the print preview, and edit printing options.

After the finite state automata is built, the user by introducing a string will be able to check, whether it is part of the language that the regular expression describes (gets accepted by the automata) or not. The whole process will be displayed, the same way as the construction process, and will have the same functionalities.

2.3 User Characteristics

Intended users of the system are university students and lecturers involved in automata theory modules. Although there are two types of users for the system, they do not form distinct groups based on how they will interact with it and thus share requirements.

Users will be able to run automata building algorithms on regular expressions as well as determinising automata to observe the visualisation of the execution of those procedures. Options to save or print the resulting automata will also be presented to users.

2.4 General Constraints

The application is constrained by the complexity of DFA which are built as a result of NFA determinisation. The process of determinising NFA has $\Theta(2^n)$ time complexity and can result in DFA consisting of at most 2^n states (where n is the number of states in NFA). Considering that the application is intended to be run on desktops with screens of standard size, visualizing DFA for large n may not be feasible.

Saving feature of the program is limited by the amount of memory available on a computer at any given time.

2.5 Assumptions and Dependencies

It is assumed that the application will run on a system that has enough memory on the hard drive to save one or multiple images, otherwise the save functionality will not perform as expected.

It is also assumed that the computer that runs the application has a keyboard and a mouse. The interface strongly relies on those and without those the program would become unusable.

For automata to be printed, a working printer connection is required.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interface Overview

The user interface will mainly consist of a single window where all input, interact buttons, and output/visualisation take place. The user interface should be tabulated,

so that the user can have multiple strings open at the same time. The user interface should show two automata at once when determinising.

The only other windows will be Save/Open/Export as Image dialogs.

The following is a complete list of input/output features that will be on the user interface.

Type	Component Type	Name	Description
Input	Textbox	Regex Input	Allows the user to input a regular expression.
Input	Textbox	Test String	Allows the user to input a test string.
Input	List	List of stored regex strings	A list of stored regular expressions for quick access
Visual Output	Graphical	Visualisation	The visualisation window for the finite state machine.
Visual Output	Graphical	Test String result	Displays whether the test string is acceptable.
Visual Output	Text/Graphical	Current Step	Displays/describes the current step or operation currently being performed.
Menu	Menu Bar Item	New	Opens a new tab with a blank document.
Menu	Menu Bar Item	Save	Saves the current/active document.
Menu	Menu Bar Item	Open	Opens a document in a new tab from a file.
Menu	Menu Bar Item	Export	Export the current/active document as an image.
Menu	Menu Bar Item	Print	Print the current/active document.
Input	Button	Play/Pause	Toggle button for playing or pausing the animation.
Input	Button	Back Step	Take a step back in the animation.
Input	Button	Forward Step	Advance a step forward in the animation.

Input	Button	Determinise	Skip to Determinising.
Input	Button	Auto-play	Button to toggle auto-play.
Input	Button	Store Regex	Store the current regular expression in the in the regex list for quicker access later.
Input	Scrollbar	Animation Speed	Sets the animation speed.
Input	n/a	Drag to Move	The user can drag with the mouse inside the visualisation window to move the FSA around the window.

3.1.3 Hardware Interfaces

The system will use a mouse (for interacting with the GUI), a keyboard (for inputting into text boxes, etc.), a screen (for visual output of the GUI and visualisation) and a printer.

3.2 Functional Requirements

3.2.1 Inputting a regular expression

3.2.1.1 Introduction

The user should be able to input any regular expression.

3.2.1.2 Input

The following characters should be used:

- Vertical bars (|) which separate alternative elements
- Parentheses which indicate the scope and the precedence of the operators
- Other characters - the user should be able to input other characters, which will constitute the alphabet of the language represented by the regular expression
- Asterisks (*) which indicate zero or more occurrences of the preceding element

3.2.1.3 Error Handling

The user should not be able to input a bad-formed regular expression. An appropriate message should be displayed to inform the user that this is not a well-formed regular expression.

3.2.2 Converting a regular expression into an automaton

3.2.2.1 Introduction

The regular expression inputted by the user should be converted into a nondeterministic finite automaton using Thompson's Construction Algorithm.

3.2.2.2 Processing

The conversion should be displayed step by step so that the user can see exactly how the algorithm works. The user should be able to specify the speed of the animation and to rewind it.

3.2.2.3 Output

After displaying the animation, the resulting nondeterministic finite automaton should be left on the screen.

3.2.3 Determinisation

3.2.3.1 Introduction

The user should be able to tell the program to determinise the automaton.

3.2.3.2 Processing

An animation should be displayed showing step by step how the automaton is being determinised.

3.2.3.3 Output

The resulted deterministic automaton should be left on the screen.

3.2.4 Saving and loading automata

3.2.4.1 Introduction

The user should be able to save and load different automata using a list provided by the interface.

3.2.4.2 Output

The automaton should be saved onto or loaded from the list.

3.2.5 Printing an image of the automaton

3.2.5.1 Description

The user should be able to print an image of the automaton.

3.2.6 Viewing the automata

3.2.6.1 Description

The user should be able to explore the automaton after animation. The user should be able to add more tabs in the window to convert other regular expressions into automata without using the first one.

3.2.7 Checking if a word is accepted

3.2.7.1 Input

The user should be able to input a word in order to check if it is part of the language.

3.2.7.2 Processing

An animation should be displayed showing step by step how the word is being put through the automaton.

3.3 Use Cases

3.3.1 Use Case 1

Name	Convert regular expression to automata
Initiator	User
Goal	Input a regular expression and have the system convert it to an automata.

Main Success Scenario

1. User types in regular expression.
2. System interprets regular expression.
3. System draws nondeterministic automata, with ε moves (with animation).
4. System determinises (with animation).

Extension

1. User presses the ‘load’ button.
    1. System fetches and presents all currently saved regular expressions.
    2. User clicks on a regular expression they want.
    3. System loads the chosen regular expression into the input box.
2. Regular expression is invalid.
    1. Ask user to re-enter a valid regular expression.

2. Resume 1.

3. The user unticked the ‘auto-play’ button on the start screen.
    1. System stops performing actions and waits for user input.

3.3.2 Use Case 2

Name	Save the current regular expression for quick access later.
Initiator	User
Goal	Save the regular expression for quick access later.

Main Success Scenario

1. User presses the ‘save’ button.
2. The system presents the save slots to the user so they can choose where to save it.
3. The user selects a slot.
4. The system saves the regular expression string in the specified slot.

Extension

3. The slot already contains a saved regular expression.
    1. Alert the user that the slot is already in use.
    2. Allow the user to choose whether they wish to overwrite it or not.
4. The system attempts to save the data but no save file is present.
    1. The system creates a file.
    2. The system writes the data to file.

3.4 Non-Functional Requirements

3.4.1 Performance

The system should provide a smooth experience. From what we have seen online, efficient algorithms can be written at least for doing generating the initial automata and determinising. Therefore, the only part of the system that will take up much of the user's time will be the animations. All background processing should be done extremely quickly so the user can be taken straight into the visualisation.

It is worth considering that to achieve processing the algorithms without disrupting user experience. They are done in separate threads, allowing the system to pre-generate all of the moves the animation shall do. Keeping a history of this is

important too, as we want the user to be able to speed up, pause and slow down the visualisation dynamically at any time. Therefore, having an efficient system is critical.

Furthermore, memory usage is not a large factor. Most computers have large memory resources and (with more help from Java's garbage collector) using lots of memory should not compromise the performance, which is useful as part of achieving the efficiency described above will probably involve storing a lot of actions in memory.

The graphics should be smooth and the interface constantly responsive not only based upon efficient computations but also in terms of the power of the system it is running on. The drawing should be easily rendered so less powerful presentation computers or devices will have no issue showing it so that it is accessible to everyone.

3.4.2 Reliability

The system should remain consistent and produce accurate results all the time. It should be consistent enough so that if a user submits the same regular expression multiple times, the exact same automata, from an identical build process, should be formed. The program should not crash and should scale to fit any window size.

3.4.3 Availability

The system will be completely offline (there will not be a licensed login or any DRM checks) meaning if the user has the program (and an up-to-date version of Java), they should be able to use it without any issues. Given it is reliable, the user will always be able to use it.

3.4.4 Maintainability

The program should be built well to allow easy implementation of new features. This may be achieved through the heavy use of interfaces. To support this, a suitable way to test the program should be fully implemented so that any new features can be validated and then committed to the complete build of the system.

Everything from design to final implementation should be well documented (including Java-Doc) so that whoever works on the system should be able to easily understand what all of the components do and where or how they should implement their new feature.

3.4.5 Portability

The system is to be written in Java, so should in general be compatible with any platform that supports Java. The main targeted systems are Windows, Mac and Linux which should be easy to make the system run on.

4. Change Management Process

The SRS will be updated constantly. Changes can be submitted by any member of the group and they will be visible immediately to all members. Changes will be discussed and approved or rejected at group meetings by means of consensus.

5. Risk Analysis

Scales will be used here and work as follows:

- Probability: 1 (low likeliness) to 5 (very likely)
- Effect: 1 (minimal effect/disruption) to 5 (large effect/disruption)

Risk	Probability	Effect	Strategy Type	Strategy
Computer malfunctions	2	1	Avoidance plan	Ensure each team member can quickly switch from using their personal machine to a lab machine and vica-versa so if one system fails the team member can continue to work.
SVN connection issues	5	3	Contingency plan	If a team member (or all) cannot access the SVN repository, the team shall share code via other media and continue to work until it is restored and the team can all re-synchronise together.
Illness / unavailability to work	3	4	Minimisation strategy	For each piece of code, at least 2 people should fully understand it, so work can continue. Also, ensure everything is committed to SVN so everyone can access it.
Compatibility issues on various operating systems	1	3	Avoidance plan	Use Java which is can be used on almost every system, and use a few external libraries as possible to minimise risk of incompatibility. We should build in the newest version of Java, but bear in mind some more advanced functions

				may not work in older versions of Java.
Underestimation of length of various components	2	4	Impact reduction	Complete the project production schedule in order and only move on when a previous item is fully complete. Only commit working and tested code so that if time runs short then there is always something to show.

6. Project Schedule

A Gantt Chart is shown below. Green squares are for development time, blue is for testing or review, red is deadlines.