Team Project - Team C5

Final Report

24.03.2016

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

Table of Contents

Introduction

The aim of the project was to create a piece of software that conveys knowledge about and demonstrates inner workings of an algorithm through computer generated visualisation. The choice of the algorithm(s) was free and thus, the decision to visualise transforming regular expression into finite-state machine of type «acceptor» was made. It was based on several factors. Firstly, as every member of the team was studying certain aspects of automata theory as part of Models of Computation module, we were familiar with regular expression based automata and knew that algorithms for generating such state machines existed. However, since creating a state machine equivalent to a given regular expression is quite intuitive, most students taking the module were not using any of the available algorithms and did not possess explicit knowledge on how they work. This presented an opportunity to create a learning tool which would familiarise students such as us with various algorithms for converting regular expressions into automata. Secondly, since finite-state machines are essentially graphs (states connected with transitions), they are naturally fit for visualisation.

The concrete algorithms that are implemented and visualised in the scope of the project are Thompson’s construction and powerset construction.

Thompson’s construction builds a nondeterministic finite automaton (NFA) that represents the same language as a given regular expression using a set of rules, each assigning a template-like NFA to one of the basic regular expression patterns (concatenation, disjunction, Kleene star and single character).

Powerset construction takes a NFA and turns it into a deterministic finite automata (DFA) by combining single NFA states into sets that represent states of DFA.

The finished program allows user to enter regular expression which will prompt step by step generation of NFA. For visualisation of this process top-down approach was picked: initial automaton is represented by two states (starting state and accepting state) connected by a single edge that encompasses regular expression in its entirety; at each consecutive step, the outermost component of the expression is expanded into its corresponding Thompson’s construction template until all edges are either empty (epsilon), or single-character transitions. Sections of NFA which are being expanded in any moment are highlighted. Upon building NFA the program presents user with an option to determinise it. Selecting said option launches visualisation of powerset construction. At this stage both NFA and DFA are displayed. As the program follows available in NFA edges and combines its states into closures, new closures are added as states to DFA. Highlighting is utilised again to stress which parts of NFA and DFA are processed at every step.

Apart from visualising the process of building automata the software includes functionality to test a string against an automata and see if it is accepted by the its language (process is visualised) and other features such as colour blind mode, saving and loading regular expressions as well as printing NFA or DFA at any stage of generation.

The following report describes development process in detail including key class structure and interface design concepts, applied software engineering techniques, test results and general work order.

Design

Description

The original specification formed a basis for how the system would work as far as the user could see. The design documentation details how the system is actually implemented and for all of the classes, including the GUI classes, UML class diagrams have been created and an overall package diagram has also been generated. The diagrams can be seen in the appendix.

Furthermore, it is based on the modified model/view/controller (MVC) style of system architecture, but changes it slightly. The model is the automata (all backend work) and the view and the controller are the visualisation and GUI.

Modules

GUI - Main modules

The view and controls are held within this main GUI modules. The view takes the form of the visualisation, where the drawing of the actual graph of the automaton is done in the canvas module discussed later. The controls are the menu bar at the top and the set of controls at the bottom. We also implemented a tabbing system so the user can have multiple instances of automatas at once, doing different things (for example one tab could be showing generation, with another tab showing testing a string).

An important part of this structure is the Document interface. It defines the minimal functions a tab needs to have to work. This is mainly applicable to the document class which holds a NFA animation followed by a DFA animation and a test string document. There are also documents which simply hold the ‘About’ and ‘Help’ HTML pages which are displayed to the user in a web view. For the documents that show the visualisation, they add to the tab the canvas, controls specific to the type of document, and the playback controls (for playing animations).

The controls specific to the documents just allow the user to input a string and push that into the automaton processor. They do have other features including keeping track of what the state of the automaton is and what should be drawn. An important part of these classes are that they run the automaton processor in another thread so as not to slow down the main interface. This also means we need to run all GUI updates in a JavaFX ‘run later’ thread to make the program thread safe.

The playback controls object has control over what frame is passed from the animation to the canvas to draw. When playing, it uses a new thread to wait, depending upon the user set speed, and move on to the next frame (or previous if in reverse). It can also be used by the user to skip frames, or skip to the end of animations in their entirety.

The menu bar itself is a class which holds all of the buttons presented, and has control over the tab list as it needs that for its operations (such as opening, closing etc.). With the tab list, it keeps a document list, where a document is something keeping track of what the user is doing in the tab and supplying resources to it. It is important for this object to also keep track of the currently selected tab, so that it can run ‘save’ or ‘test a string’ on what the user is currently looking at.

Also accessible via the menu bar are ‘About’ and ‘Help’ pages. These open in new tabs and either give the user our logo and names, or help with how to use the product. It is enforced by the main menu manager that a maximum of one of each of these documents are open at a time. The application switches to the currently open one if the user tries to open another.

An important feature of the main menu is also a ‘toggle colour blind mode’ button. This sets a flag (initially false) and sends the update through to the canvas, which uses the flag to switch between normal colours and colourblind colours when drawing the automaton and the text on the screen. To choose the colours used, the team spoke to a colour blind user to help determine the best selection of colours to accommodate for the maximum number of colour blind people (as there are several types of colour blindness).

GUI - Canvas

We decided to give the main visualisation its own module to help reduce dependencies on other sections, so it was easier to maintain and use for different types of drawings (mainly generating automata and testing string automata). This means giving control of the canvas almost completely to the components classes described above. This also effectively forms the view of the MVC model.

The main class in the gui.canvas package extends a normal JavaFX canvas, so it is easy to change information about and add our own methods to. Alongside this, we created a class purpose built for holding a ‘frame’. A ‘frame’ is one state of what we want to draw on screen, holding information about positions of nodes and edges and a few other small bits of information including the text displayed at the bottom of the screen.

The information about nodes and edges was stored in their own two classes, VisualState and VisualEdge, so that it was easier to build the frame (it is a collection of objects), but also so we could give them custom draw methods where we just pass the objects the canvas we want to draw on. Furthermore, we also store information in the nodes and edges about the highlighting and the type of state for nodes and the type of curve for edges for use when drawing.

Using the above structure, it is easy to perform a draw routine. The canvas initially does some calculations, with information passed from the components, to set up sensible scaling and sizing of nodes. This mainly takes into account the size of the screen the user is using along with the number of nodes in the automata. It then goes through and draws the graph, edges then states. It draws them in this specific order because the edges extend into the middle of the states, so by drawing the states after the edges, we ensure no objects are drawn over each other.

Automaton

Under our revision of the MVC, this forms the core model. The automaton package contains everything related to interpreting user input, generating the automaton from that and processing it into frames of generation. It will them deliver that to the GUI section.

There is an automaton class which holds all the information about the automaton we need to perform calculation, and an animation class which holds basic information about the frames of animation which is extended by an NFA, a DFA and a test string class. These animation classes use information about the automata to generate their frames showing their respective purpose. Once they have finished processing the information, they return the frames to the GUI module which displays a certain frame.

Moreover, this means that the automaton module handles the graph layout. For the prototype and early tests (before a custom one was written by the team), we used a library called JUNG which would lay the graph out. It gave us the knowledge that our graph was being generated correctly in terms of the edges and states and while we always wanted to write our own implementation for the actual layout, this emphasised the need for it even more. The key to the layout was having knowledge of the overall graph at each frame and what the final graph should look like. Our final implementation requires no external libraries and contains only code written by the team to lay out the graph.

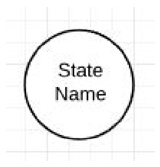
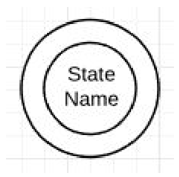
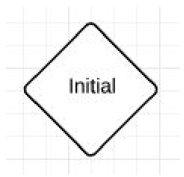
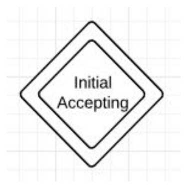
Testing

We made a package containing all the automated testing. This took the form of a series of JUnit tests to check that the algorithms were calculating the correct automata (and closures within them) and frames for the visualisation. This module, while not being core to the functionality, is vital to get correct output from as it helped us determine whether the program runs correctly and make the whole system more reliable and easier to maintain while working on other parts of it. This is also worth including in the source code as anyone who would like to extend the project is able to reuse our tests and repurpose them to check their own code.

Visualisation Design and Graphical User Interface

Automata Design

For the automata we wanted out design to be simplistic, clear and to use the same shapes that we learn about in the Models of Computation module for different types of states.

Normal State	Accepting	Initial State	Initial and Accepting
			

Software Engineering and Risk Management

Overview

For such a big project as this one it was evident that we had to use some sort of software engineering techniques to plan our project, and organise our work around one or more software development models.

Techniques used

Model-View-Controller

We structured our project based on the Model–view–controller (MVC) software architectural pattern: we split our team into two groups one working on the GUI/Visualisation part (view and controller), and the other group implementing the simulation of the algorithm (model). Those two groups worked mostly independently, and worked together only when we integrated the different parts of our project together. Even though we worked independently, we attended the meetings together, and the communication between the groups was really good, which made the integration of the different aspects really easy.

Agile

We decided to follow the agile software development model. We split the whole project into small independent functionalities (product backlog) and we estimated how long is it going to take to implement each functionality. We prioritized them based on how important they are, and how fast can we implement them. We had scrum meetings at least once every three days (every day during last week), where we discussed new ideas and selected the functionalities (sprint backlog) that we are going to implement next.

The agile model we used is an iterative software development process, at each sprint we designed and implemented functionalities, and also tested the old and those newly implemented functionalities as well, to make sure everything works fine together.

One of the benefits of using this software development model, is that we built up an application really fast (at the end of our first big sprint), so we could easily have our prototype ready by week 6. Also, using agile makes it easier to implement any change in the requirements document, at any stage. This way we can easily adapt to the received feedback, we can change the requirements without having to change everything in our project.

Pair programming

Every time we implemented a more complicated functionality we worked in pairs, to make the implementation more optimal, but also to reduce risks: to reduce the bus factor. This guarantees that for each functionality there are always at least 2 members from our team who have full understanding of the code. This ensures us that even if one of our members (for any reason) cannot work on the team project anymore, the other members would be able to continue the project, use or even edit any of the already implemented functionalities.

Risks

Evaluation

In order to evaluate our project we compared the final project results against the requirements and features stated in the Software System Requirements.

From the SRS document it shows that our system purpose was to develop a learning tool to help Computer Science students learn about Finite State Automata and understand the algorithms used in building a deterministic finite automaton by visualizing each step of the construction. In section 2.2 of the SRS, the “Product Function” states what functions our app must have. This included typing in a regular expression, checking if the input is valid or not, watching how the automaton is built, being able to pause and manually go through each step, print or save as image, zoom in, zoom out and check for bisimulation. The functions we did not implement were minimising, zooming in and out and checking for bisimulation. Apart from these, all of the proposed functions were fully implemented.

We planned to make our app to be user friendly for our targeted audience. We focused on adding H.C.I features like creating a help page for the users. On the help page, the users can see a general description of the algorithms involved in building automata with links to further information about the algorithms and a tutorial on the buttons functionality and how to switch from a stage to another.

As part of our testing we chose to do User Testing which included giving our app to users which met our target audience. We first had to reach a final GUI concept and ensure that most of our functional requirements were implemented. We designed a questionnaire to cover all the important aspects of our learning tool and analyse the results. Once the results of the questionnaire from the users were collected, we reviewed their suggestions in detail and grouped them based on their relevance to our application’s functionality. We have managed to receive feedback from students and two lecturers that are familiar with automata theory. We even tested our colour blind mode on a colourblind person to make sure that it is properly visible to people that require special arrangements to use our application. The high priority suggestions have been fully implemented in order to meet potential users’ requirements. Some of the suggestions we received and considered important and immediately were as follow: adding a help page, adding a title for the current phase of the algorithm, adding a speed slider and showing what character the algorithm is being searched next in the automaton. Unfortunately, there were still other suggestions but we considered that they would just extend the current functionality of the program and would not make a serious impact on the user ability to understand the algorithms. Overall, 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.

Since we were not allowed to use external libraries that can help with our graph representation, we have focused on not using any extensive UI libraries except for JavaFX. We have followed this constraint and have solely used standard Java and JavaFX to complete our project. This made the project more difficult but the challenge was overcome by implementing our own graph representation.

All functional requirements have been met except for bisimilarity and minimisation. The reason why we could not implement these functions is because we did not take into consideration the time and complexity of building an automaton from a regular expression and testing if a word belongs to that automaton. Another issue was to calculate the position of each state so that they can maintain their position at each step in order to make the whole automaton more clear to the user. This issue along with calculating the scale factor proved to be a serious challenge as the layout of the states on the screen is an important factor in making the user understand the algorithms steps.

All the non-functional requirements have been met. The performance of the application enables the user to have a smooth experience with a quick response time to user interaction. In terms of reliability, the application builds the same automaton when the same regular expression is inputted and it always scales itself based on the window size. In terms of portability, there were no errors encountered as the application works correctly on all platforms. Availability and maintainability are all also as proposed in the specification. The application is available for any user who has an update-to-date version of Java without any requirements. In terms of maintainability, our application's code is well structured and easy to read so that the system can adapt to new changes without affecting other features of the program.

We believe that our application turned out to be as we planned it to be, without changing the idea of how our program would look like. I think the most impressive features of our program are the graphics and the quick computation of the frames of the animation. These tasks were successful because we started early and we had time at the end of our project to improve on our design and backend algorithms of constructing the automata. Other reasons why we believe we have done a great job are teamwork and our organisational skills. We managed to work on our strengths and divide work equally.

If we were to start this project again, the main thing that we would do would be to have a more realistic approach of the planning stage. We overestimated the amount of time some of the tasks require and we thought we could reach all our goals for this application but a more accurate Gantt Chart would have helped stay on track with our work. Furthermore, we would focus more on early user testing because it would help with our application design in advance.

In conclusion, we consider that the project can be evaluated to be an overall success as we have met the healthy majority of the requirements set in the SRS document at the beginning of the project.

Team Work

General Organisation

The organisation of work in a team is one of the most important aspects. It is crucial to allocate pieces of work to different members appropriately and establish a certain pace of making progress. Without that, it is impossible to make a certain amount of progress in time and coordinate the development of the project appropriately.

During our project, we agreed on having three meetings per week. That was enough to do a significant amount of work every week and not too much for us to fall behind with other coursework. At those meetings, we were discussing ideas that we came up with and problems that we encountered as well as assigning new tasks to people, merging our code and making sure it was compatible.

At the very beginning, we decided that it would be best if we split into two teams. It would have been quite difficult to work on every task as six people and that would not have allowed us to make significant amount of progress. One team was dealing with the GUI interface and the other one with implementing the algorithms. On the one hand, that was very useful in keeping one thing separate from the other, but on the other hand it was hard to keeping everything compatible. Nevertheless, we take measures to maintain all the components coherent and succeeded in doing so. Both teams were constantly communicating with each other, exchanging ideas and asking what the other side of the project looked like. Everything was adjusted continuously so that it matched the other team's code.

Our decisions about who was going to do which task within one team were based on the suggestions and the ideas that each member had and skills and good understanding of each of us. At the start, one of us suggested to use the JavaFX library and that person was appointed to start doing research on it and implementing it into our project. Later on, other people turned out to have an excellent understanding of one of our algorithms, therefore we decided that they would start its implementation.

We have been using pair programming. As it was a highly complex project, we would always make sure no one programmed on their own to reduce the risk of various components being incompatible with each other. It also really helped us work as a team rather than individually and was beneficial in terms of sharing ideas and remembering about tasks that still needed to be done.

Problems, conflicts and solutions

The more complex project you are involved in, the more problems and conflicts you come across and some of them are inevitable. Although, the important aspect is not how many problems you encounter, but how well you are able to manage with them.

The main issue that we had cropped up at the beginning, when we were to decide whether to use a top-down or bottom-up approach. The top-down approach creates one automaton which is constantly expanded, whereas the bottom-up approach creates a number of small automata and joins them together. We had a lot of debate about both of them and we considered several pros and cons, but eventually, by means of consensus, we decided that the top-down approach would be better as it is clearer and more readable to the user. We thought that it would be easier to focus on one automaton and its expansion instead of having a number of them scattered around the screen.

There were also a number of other, minor, problems, such as different methods having to be adjusted to be more compatible and universal. For example, we created a method determining the coordinates of an NFA on the canvas and later on it turned out that we needed the same method for a DFA. We did not predict that at the beginning, therefore the previously mentioned method had to be reimplemented.

In general, we have managed to overcome every difficulty and none of them set us back to a significant extent.

Communication

It was crucial to keep in touch with everyone at all times so that everyone knew what was happening with our project, what needed to be done and when we were going to have next meetings. There were a number of tools which turned out to be extremely useful for us in efficient communication.

Google Drive

All of our the documentation, design, plans and reports were stored on Google Drive the whole time. The documents could be edited by more than one person at the same time so everyone could see the changes made by others in real time. Everyone had access to everything and it was easy to keep all the files neatly in one place.

Facebook

We were also using Facebook chat where almost three thousand messages have been sent in our group conversation. It really helped us get through to each other at any time and all important issues, such as new ideas, problems or planning future meetings, were always discussed there. Communication would have been much harder without it; thanks to that tool, we practically never fell out of touch with each other.

Skype

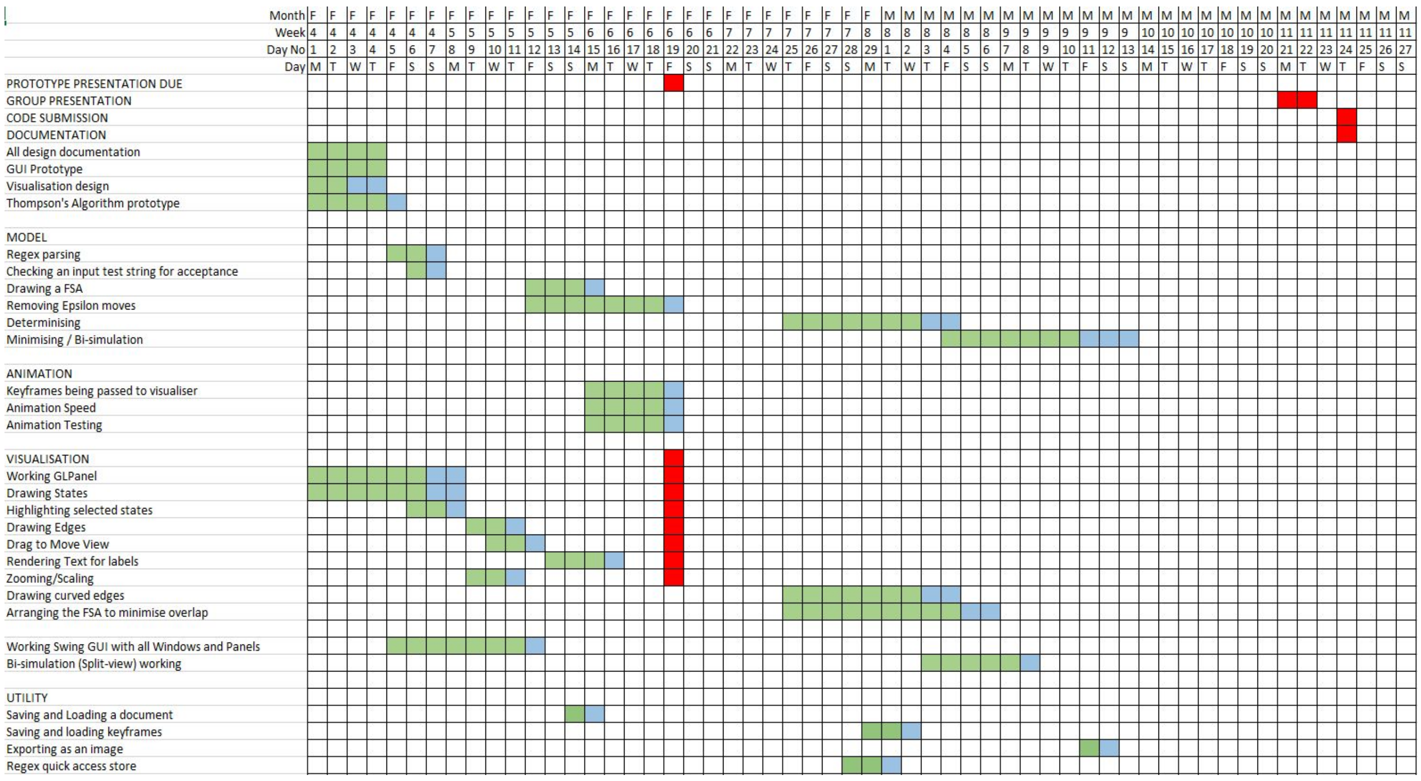
Sometimes we were not able to meet as some of us were away. An extremely useful tool is such situation turned out to be Skype. By means of it, we were able to have a discussion face to face without having to go out which was very convenient.

Gantt Chart

We designed a Gantt Chart which can be seen in the appendices and planned all the work with one day precision. It was extremely useful as an overview of all the tasks that needed to be dealt with and all the stages of the development of our application. However, it turned out that not everything was able to be done according to the plan. That was because some of the components required more time than we expected at the beginning. There were certain problems that occurred later and that we were not aware of before. There was also other coursework that we had to keep up with and sometimes we did not get as much time to devote to our project as we planned.

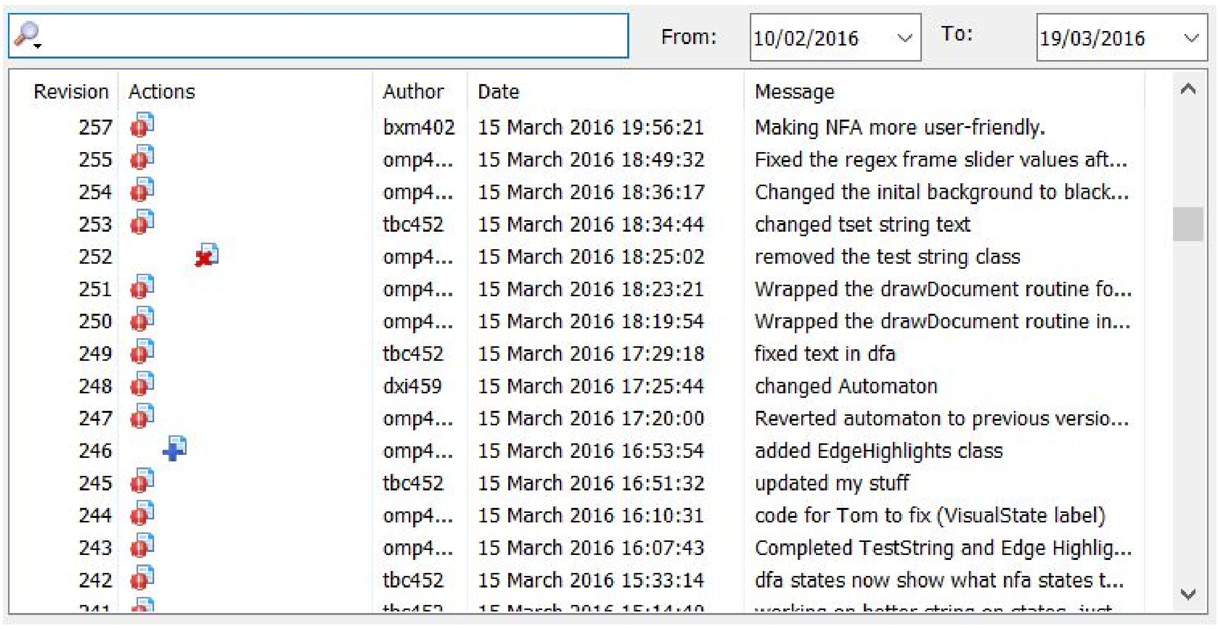
Nevertheless, thanks to a few mini scrum meetings that we organised, we always avoided falling behind with the project and managed to get the tasks done in time. At those meetings, we would spend a lot of time on intensive programming and we would not finish until the desired feature had been implemented.

Our Gantt chart can be seen below:



SVN

We have been using SVN where our code was stored at all times. We were constantly updating all the changes to it so that each team member could download the latest version in any place at any time. That really helped us keep the whole project compatible and neat. A fragment of our SVN log can be seen below:



From: 10/02/2016 To: 19/03/2016

Revision	Actions	Author	Date	Message
257		bxm402	15 March 2016 19:56:21	Making NFA more user-friendly.
255		omp4...	15 March 2016 18:49:32	Fixed the regex frame slider values aft...
254		omp4...	15 March 2016 18:36:17	Changed the inital background to black...
253		tbc452	15 March 2016 18:34:44	changed tset string text
252		omp4...	15 March 2016 18:25:02	removed the test string class
251		omp4...	15 March 2016 18:23:21	Wrapped the drawDocument routine fo...
250		omp4...	15 March 2016 18:19:54	Wrapped the drawDocument routine in...
249		tbc452	15 March 2016 17:29:18	fixed text in dfa
248		dxi459	15 March 2016 17:25:44	changed Automaton
247		omp4...	15 March 2016 17:20:00	Reverted automaton to previous versio...
246		omp4...	15 March 2016 16:53:54	added EdgeHighlights class
245		tbc452	15 March 2016 16:51:32	updated my stuff
244		omp4...	15 March 2016 16:10:31	code for Tom to fix (VisualState label)
243		omp4...	15 March 2016 16:07:43	Completed TestString and Edge Highlig...
242		tbc452	15 March 2016 15:33:14	dfa states now show what nfa states t...
241		tbc452	15 March 2016 15:14:49	working on better string on states, just

Summary

Based on massively positive user feedback and overall test results it can be concluded that the software produced meets its requirements when it comes to comprehensively visualising regular expression to NFA/DFA transformation. We believe that our program can be used to complement automata theory related courses due to its ability to quickly create and neatly display various automata of a certain type, or as a standalone tool for learning Thompson's and powerset construction algorithms, since visualisation along with descriptions provided in the «Help» section of the application should be sufficient for a user familiar with automata in general, but not with concrete algorithms, to gain working knowledge of those algorithms.

We believe that workflow was well-structured and organized throughout the project with all team members contributing and providing valuable input. Design decisions, however, could be thought out more thoroughly in the early stages of development, since the team ended up making some significant design adjustments later on. Although one of the most significant design decisions made early (top-down approach to visualisation over bottom-up) that defined how the whole application functions we reckon to be the correct one as it allowed us to gradually incorporate changes into automata while preserving early established general structure and building up on it. The choice of agile development paradigm proved to be a good one too since it helped us to stay flexible and quickly introduce decisive changes while keeping everyone up to date on the current state of the project.

Another thing that could admittedly be done better is incorporating automated unit testing more deeply into development as we repeatedly reintroduced already fixed mistakes into the code by accident on a number of occasions, mostly through SVN.

Individual Summaries

Thomas Clarke

Throughout the course of team project, I held an important and authoritative position. I was mainly teamed with Owen to produce the GUI and visualisation, of course while working with the whole team to make sure the different components actually fit together well. Myself and Owen put many hours into developing the visualisation and iterating upon it until it looks as we wanted it to, along with a user friendly GUI with a host of easily accessible features.

A big part of the experience for myself was being a key member of organisation and in particular managing team meetings and trying to drive the content of conversation at them. It was important from creating the original specification to submitting the final product that our group's ideas were synchronised and effectively used. For instance, I suggested and set up the Trello board we used to help keep track of current tasks and notes associated with them, so whoever wanted to work whenever could access as much information as possible (along with all other documentation hosted on Google Drive).

In terms of programming, I suggested the use of JavaFX as a good platform. Despite all of us being inexperienced with the library, I created the first mock-up of the GUI (while Owen created the first drawing) with in a few days of the suggestion to prove its potential. I then went on to program many features of the GUI and visualisation, including (but not limited to) the playback controls, handling multiple animations and the file menu (save, open, print and close). I also found and worked with the colour blind user to help develop 'colour blind mode' to help the software be as accessible to as many people as possible.

I also worked with other team members to help smooth the connection between the backend and the frontend, namely the Animation class and the Document interface. Towards the end, I also did a lot of work in refactoring the project to make the GUI as modular as possible with as low dependencies as possible, so that features (such as the side bar with DFA) were easier to add, and providing appropriate JavaDoc throughout all GUI classes.

I feel as though I worked appropriately hard for the project, not over contributing and keeping to deadlines. I enjoyed learning JavaFX and working in this team environment to produce this large piece of software.

Mihnea Patentasu

I believe that working on the team project has been an overall enjoyable experience. My main role within the team was as a backend developer for our application. I worked on implementing Thompson’s algorithm that is being visualised by the program, but I also worked on the Testing Strategy by creating unit tests for our base classes.

I am confident that I have learnt a lot from this module, both academically and as a person. On the academic side, I learnt more about different software engineering methods and how to work with SVN. I have understood that you cannot just start coding, software projects require planning and documentation. My coding skills have improved, as well as my research skills. Our chosen class of algorithms was not implemented in Java so we had to research and analyse which data structure would be best suitable for our algorithms in order to reduce complexity. Furthermore, I have also understood the importance of refactoring your code so that your team members will not struggle to understand your code. Another useful technique I have learnt was testing which helped us find bugs in our code easily whenever someone has made changes to the code. Using Trello has proved to be a useful way of organizing and grouping all our tasks. I am sure that these skills will help me further in my studies and even in the industry.

Working in a team that I did not choose was an interesting experience but I had the luck of finding myself in a skillful team. At first was a bit difficult as we had to adapt to the different styles of programming and we were not used to working with each other. We did not know our strengths or weaknesses. But that changed after a few weeks where we decided on our application’s design. So we quickly went from forming to storming. I have also learnt how to effectively work well within a team to achieve a common goal before a deadline. I enjoyed employing pair programming because we could share ideas easier and work more efficiently.

If I could repeat the whole project, I would probably do more tests before coding and try to do more pair programming as I believe it generates better, safer code and sometimes you do more progress when there's someone from your team pushing you and working alongside you.

In conclusion, I believe that my team and I have worked hard to produce an innovative practical software that can help students understand automata theory. I think that we should be proud of our product and the short amount of time that we managed to complete our work.

Piotr Wilczyński

Working on a project with team C5 was an absolutely amazing and unforgettable experience. I have really enjoyed it and I am extremely proud of everyone's attitude and contribution. The fact that I have improved a number of skills, such as effective communication, planning and creativity, will undoubtedly be very beneficial for me in the future.

During the development of the project, I was part of the team implementing the algorithms. I contributed significantly to the initial discussions about what the interface should look like and how certain objects should be represented in the code. I was one of the people who implemented the method for string testing. I wrote a lot of unit testing to always make sure that the whole project was still working. Moreover, I helped with designing the questionnaire and collected feedback from several people who are also studying the module related to the automata theory.

However, my main role in the team was to coordinate work and keep track of what needed to be done. I was always making sure that everyone had something to do and trying to balance workload. A significant number of components, such as implementing classes, determinisation and testing strings, were kicked off by me. My task was to monitor the amount of progress we were making and ensuring that we were not falling behind. I was the one to reflect on what had been done so far and remind everyone about the things that had not been dealt with yet.

One of the most significant aspects that I brought to everyone's attention was error handling. I was aware that it was very important for the application to deal with incorrect inputs appropriately. I implemented a method which checks if a given regular expression is well-formed and well-bracketed. I also suggested to the GUI team that they showed an error message when a wrong regular expression had been entered.

Apart from that, I made a significant contribution to refactoring. I was always watching the whole code, reviewing all the changes and trying to keep the project clear, neat and concise.

To sum up, I am extremely happy that I had a chance to be a member of team C5 and I think we have managed to produce a decent piece of software. We devoted a huge amount of time to make it an outstanding learning tool for students and we are pleased with our results. It has been a great pleasure to be part of this project and I think everyone had benefitted from it to the fullest.

Danyil Ilchenko

Owen Pemberton

Botond Megyesfalvi

References

Appendices