COMP390

2020/21

Automata Tutor Extension

Student Name:	Jan Clare
Student ID:	201319103
Supervisor Name:	Dominik Wojtczak

DEPARTMENT OF
COMPUTER SCIENCE

University of Liverpool
Liverpool L69 3BX

Acknowledgements

I would like to thank my parents Anthony and Hedvika as well as my Nana Rosalind and Grandfather Joe for supporting me throughout my studies.

Thanks also to Mr Joseph Livesey for being a fantastic teacher.

I would also like to thank my friend Ezzat, despite seeing which way the wind is currently blowing.

Thank you to all the wonderful people from Bogdan's House for keeping me sane.

COMP390

2020/21

Automata Tutor Extension

DEPARTMENT OF
COMPUTER SCIENCE

University of Liverpool
Liverpool L69 3BX

Abstract

Automata Tutor is an online learning tool designed to “help students learn and teachers teach” (Weinert & D'Antoni, 2021) Automata Theory and Formal Languages. The website consists of teacher set exercises for students to complete. The tasks are marked with intelligent automatic feedback.

The aim of this project was to implement at least one new problem type to extend Automata Tutor. The initial plan was to implement DFA Minimisation and Probabilistic Parsing exercises. DFA Minimisation was successfully implemented, however Probabilistic Parsing was replaced by Regular Expression to DFA since this was more similar with existing problems so patterns and code could easily be mimicked or cloned and refactored. Another factor in this decision was time constraints which is also the reason its implementation is unfinished.

In summary, DFA Minimisation is the process of eliminating states that are equivalent in a DFA to find a DFA that accepts the same language but has less states than the original. Regular Expression to DFA is the process of converting a Regular Expression to a DFA (Sisper, 1997).

This project proved to be an exercise in Software Engineering. To accomplish the aim, a good understanding of how the existing implementation worked and its architecture, as well as all technologies involved was required. Acquiring this knowledge was not trivial and became a large portion of the project since the Automata Tutor documentation is brief and this Author lacked prior applied experience of Software Engineering on a multi-tiered application this large. It was also essential to have a prior background in Automata Theory. A secondary aim of this project was to produce a dissertation good enough to be a useful document for future developers trying to implement their own problem types.

Table of Contents

Introduction and Background

This section, as well as the Theory and Technology sections will explain the research undergone and knowledge required to make a successful implementation. Ideally, this will provide enough understanding to tackle the rest of this dissertation.

Project Prompt

"Deploy at a local server and add to Automata Tutor 2.0 some additional feature."

Motivation

The number of students with an interest in Automata Theory has significantly increased over the past decade (Patterson, 2013). Teachers face an increased workload to provide personalised homework feedback. Automata Tutor was created to automate this process. Typically, the problem with online exercises is the quality of feedback is incomparable with teacher marked work (D'Antoni, et al., 2015). This is because automating high-quality personalised feedback is challenging.

Over several tests and versions of their system, the developers of Automata Tutor concluded that extensive feedback could be confusing whilst binary feedback could be detrimental to student progress (D'Antoni, et al., 2015). So simple and concise feedback was best. A Study done by the University of Pennsylvania (D'Antoni, et al., 2015), concluded that a mix of counter examples, as well as their own hint-based feedback was best.

So, in conclusion, the motivation for Automata Tutor is to reduce the workload of teachers as the number of students with an interest in Automata Theory increases. Crucially this tool needs to provide quality feedback at least comparable with teacher marked work. The motivation for this project was to improve Automata Tutor by extending it.

Aims

1. Adding at least one new problem type to Automata Tutor.
2. Adding the ability for users to create and complete exercises on DFA Minimisation with automated feedback.
3. Adding the ability for users to complete exercises on Regular Expression to DFA with automated feedback.
4. Effectively integrating these features into the existing system.
5. Gaining experience of Software Engineering on a large multi-tiered application.
6. Learning about various technologies, conventions, and coding patterns.
7. Learning about how Automata are represented and worked on with modern computers.
8. Writing a dissertation good enough to be helpful documentation.

Objectives

1. Getting Automata Tutor working in a local debugger.
2. Developing an effective workflow (compiling and running code and viewing changes in a browser to experiment with existing source code).
3. Understanding existing application design.
4. Learning to use the toolset (Git, Scala + Lift Web Framework for frontend; C# for backend; h2 database engine for data tier).
5. Figuring out how features can be integrated into the existing application.
6. Implementing a DFA minimisation problem type.
7. Implementing meaningful automated feedback.
8. Implementing a Regular Expression to DFA exercise.
9. Implementing meaningful automated feedback.
10. Testing implementations to validate they work.

Theory

Finite Automata

Finite Automata are useful for modelling computers with basic functionalities (Sisper, 1997). They consist of:

- A set of states.
- Transitions between states that depend on an input symbol.
- An alphabet of acceptable input symbols.
- A starting state.
- A set of accepting states.

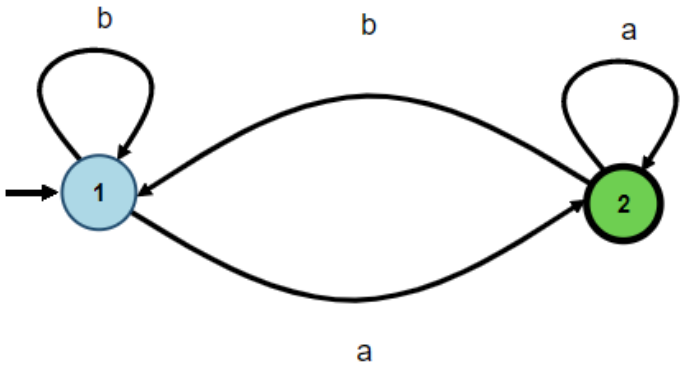


Figure 1.0: Deterministic Finite Automaton constructed using the most recent version of the Automata Tutor website: https://automata-tutor.model.in.tum.de/. (Weinert & D'Antoni, 2021)

Finite Automata can also represent languages. This is “the set of all strings that result in a sequence of state transitions from the start state to an accepting state” (Hopcroft, et al., 1979). You can see how Figure 1 bears all the properties of a Finite Automaton. There is a set of states: {1,2}. There are transitions between the states which depend on symbols from an alphabet: {a,b}. The start state is state 1 and the accepting state is state 2. Finally, you can decipher the kind of language it represents by closely examining the Automaton.

There are three types of Automata relevant to my project. Deterministic Finite Automata (DFA) means that in any state on a given input symbol, there is only one transition. This includes a transition to a rejecting state which is implicitly represented by having no transition for that input symbol. Therefore, you can determine the exact result of inputting a symbol from a state. Figure 1 is a DFA. Conversely, Non-Deterministic Finite Automata (NFA) means that there can be many transitions from a state on a given input symbol. This implies that you cannot necessarily determine the resulting transition of an input symbol from a state. (Hopcroft, et al., 1979)

Finally, there are Epsilon Non-Deterministic Finite Automata (εNFA). These are the same as NFAs, but Epsilon (ε) is added to the alphabet of acceptable symbol inputs. ε has unique properties. ε transitions are hard to define without an obscure mathematical definition so the following informal explanation will be sufficient for the scope of this project. “Think of Automata as accepting sequences of labels along paths from the start state to an accepting state. Each ε along a path is invisible; i.e., it contributes nothing to the string along the path” (Hopcroft, et al., 1979). ε transitions are special because they are like free transitions; no input is needed to transition to another state.

Deterministic Finite Automata Minimisation

Every DFA can be reduced to a minimum number of states whilst still accepting the same language (Hopcroft, et al., 1979). There is a process to minimising a DFA where you identify indistinguishable states and represent them as a single state.

After acknowledging and ignoring unreachable states, the general method to finding indistinguishable states is to create a table that compares every state pair (excluding with themselves). You then algorithmically go through each state pair that has not already been eliminated and eliminate them if they are distinguishable (Sisper, 1997). You repeat this until a cycle produces no new distinguishable state pairs. These final indistinguishable state pairs will be equivalent and the DFA can be minimised. There are multiple giveaways that two states are distinguishable. Firstly, eliminate all pairs where one state is an accepting state and the other is not. Next eliminate the states, whose

transitions on the same input, respectively reach two states in a previously discovered distinguishable pair (Hopcroft, et al., 1979).

The original plan was to implement the minimisation problem type so that students would be forced to use this method. This could have been done by implementing the table described above and users could get feedback on it. In the end, only a canvas where students could input their attempt minimised DFA was implemented. It is up to the student whether they use the standard method or not. Minimising complicated Automata without the method is extremely difficult, so a student's understanding of it will be tested on harder problems anyway.

Regular Expressions

Finite Automata are machine like model representations of languages, but they can also be represented with an algebraic description (Hopcroft, et al., 1979). There is no way to easily explain Regular Expressions briefly so this section will outline only the most relevant parts for this project, as well as some important properties. However, Regular Expressions are an extremely interesting topic and have many useful applications beyond the scope of this project. There are some excellent textbooks with more detailed and accurate descriptions of Regular Expressions, complete with formal definitions. These books have also been referenced frequently throughout this dissertation. They are: "Introduction to the Theory of Computation" by Michael Sisper and "Introduction to Automata Theory, Languages, and Computation" by John E. Hopcroft, Rajeev Motwani and Jeffrey D. Ullman. (See the bibliography for their full references).

Regular expressions consist of a sequence of characters (Friedl, 1997) of which there are two types: Operations and Terminals (Weinert & D'Antoni, 2021).

Terminal characters represent the occurrence of a character at a certain point in the strings accepted by the language.

- Any character from the language alphabet is acceptable.
- ε or epsilon represents a language containing only the empty string.
- Ø or empty set represents a language with no strings in it (Sisper, 1997).

Operation characters describe expression properties. Their format varies depending on the operation. Here are some examples:

- * is a super script and comes directly after a Terminal character or expressions encircled by brackets. It specifies the expression or character can occur any number of times including not at all.
- $^+$ is a super script and comes directly after a Terminal character or expressions encircled by brackets. It denotes that the expression or character must occur at least once and then any number of times.
- + is normally between two expressions and says that either expression can occur.

$$(a+b+\epsilon)^*a^+$$

Figure 1.1: Example of a Regular Expression.

Figure 1.1 is a Regular Expression that denotes the language represented by a set of strings. Every string in the set is of the form: any number of "a"s including none or any number of "b"s including none or empty string, followed by at least one "a".

Converting Regular Expression to Deterministic Finite Automata

"Regular Expressions can define exactly the same languages that the various forms of automata describe." (Hopcroft, et al., 1979). "Any Regular Expression can be converted into a Finite Automaton that recognises the language it describes , and vice versa" (Sisper, 1997). Hence, there is a way to convert Regular Expressions to DFAs.

The general method is to convert regular expressions to an εNFA which can be converted to an NFA which can be converted to a DFA.

Regular Expression to εNFA

Regular Expressions can be directly translated into εNFAs. As you can see in Figure 1.2 you can convert elements of a regular expression to their εNFA equivalent and piece together the result so that you have a complete εNFA. You can then simplify the εNFA in preparation for conversion to an NFA.

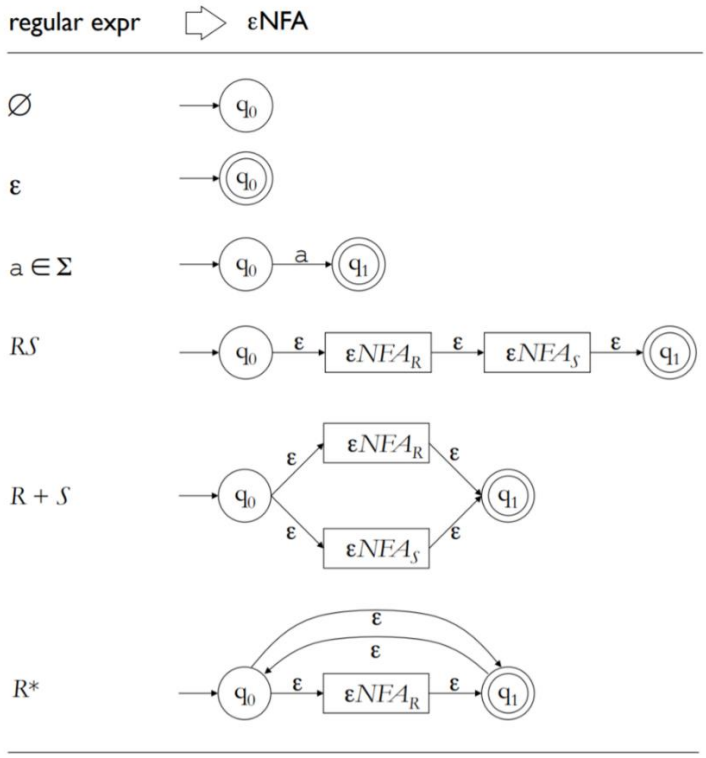


Figure 1.2: "General Method" (Wojtczak, 2019). Table of translations from Regular Expression to εNFA.

εNFA to NFA

To convert an εNFA to an NFA you need to remove ε transitions. Make a table that represents reachable states from a given state on a given input symbol. Each row should be a state in the Automaton, each column should be an input symbol, and the contents of each field should be the reachable states. This table will represent an NFA equivalent to the εNFA we are trying to convert so ε input symbols should not be included in the table. To fill a field in the table, asses which states are reachable with a transition labelled with the symbol in question from the state in question. This should be straight forward with regular symbols. To illustrate the nature of ε transitions, consider a state "X" and a symbol "p". If there is an epsilon transition from state "X" to another state "Y", and state "Y" has a transition labelled "p" to a state "Z", then "Z" is reachable from state "X" with input "p". If there is a transition from state "X" labelled "p" to state "Y", and state "Y" has an epsilon transition to state "Z", then state "Z" is also reachable from "X" on input "p". It is also worth mentioning that any amount of ε transitions accompanied with an input symbol somewhere can be used to reach another state.

Once the table has been filled, you have the means to draw an NFA with no ε transitions. Draw the states with their new transitions from the table. Any states that had an ε transition to an accepting state in the εNFA is an accepting state in the newly constructed NFA.

NFA to DFA

To make this conversion, you must also draw a table. This will be used to build a DFA. The method is similar, but it has its own nuances. The format of the table is the same, i.e., each row should be a state in the Automaton, each column should be an input symbol, and the contents of each field should be the reachable states. This time, the table is constructed as you fill it in. Firstly, make a row for the start state and fill its fields with the states reachable on transitions with the given input symbols. Instead of each state in the field persisting as an independent state in the new DFA, each field containing a new combination of states becomes its own state in the DFA, which you now have to give a row and assess in the same way as before. Repeat this process until there are no new fields with a new combination of states i.e. there are no new states to add to the new DFA. Any state formed by a field which contained an accepting state from the NFA becomes an accepting state in the DFA. Now you are left with a table that gives you a map of the new DFA. This DFA will be equivalent to the initial NFA.

Automata Tutor

Originally there were plans of implementing tables in the User Interface to be filled when a student attempts to solve a problem. This would prove students understood the method described above. However, the same conclusion as with DFA Minimisation was reached. It was better to keep the User Interface simple. Students would probably use this method to solve trickier problems anyway and it also seemed restricting to force students to use it. Furthermore, it would have taken considerable effort to implement a table functionality from scratch, whereas utilising the existing Automata canvases was more straight forward.

Technology

This section will explain the technology used and discuss their role and significance in Automata Tutor.

Git and GitHub

Git is a distributed version control system (Long, 2008). GitHub is an online platform for software development which uses Git (GitHub, 2021). GitHub is the home of many open-source projects like Automata Tutor. The Automata Tutor repository is where the initial code was accessed from. It also contains a Read-Me document which was the starting point for learning about Automata Tutor's implementation.

A more up to date version of Automata Tutor exists (version 3) with many more problem types but has not been made available on GitHub yet. This project was to extend the available version (version 2). Furthermore, the problem types implemented in this project are not implemented in the latest version of Automata Tutor.

Multi-tier Application Architecture

Multi-tier architecture is common in software system design. It physically separates presentation, backend logic, and data storage. There are many advantages for this type of architecture including scalability, reusability, security, and manageability (Voth, et al., 1998). This architecture is used for Automata Tutor, which consists of the following components:

- Front-end web site, which provides both the presentation tier and middle-tier business logic.
- Database.
- Back-end web site, which implements the grading engine.

Model View Controller (MVC) Pattern

MVC is a coding pattern that is often used in applications that have a User Interface. Similar to multi-tier architecture, this also separates presentation (view) from data (model) and business logic (controller) (Code Academy, 2021). In Automata Tutor there are Lift templates (the views) which present the model in its current state to the user. There are Scala snippets (the controllers) which controls the model for viewing. Finally, there is an object relational mapper which takes objects like "Problem" and "SolutionAttempt" types and persists them to a database (the model).

Lift Web Framework

Web Frameworks are code libraries that simplify the task of building web applications (Wikipedia, 2021). They do this by providing abstractions for commonly performed tasks. This reduces the amount of code that has to be written overall, and it also encourages good design (because each framework tends to support specific ways for structuring web projects and handling web requests).

Automata Tutor's frontend is built with the Lift Web Framework. Lift is probably not the most popular web framework for Scala today. Internet searches for "Scala web framework" often put Play and Akka, for example, ahead of Lift. Lift was probably chosen because it was a popular choice when Automata Tutor was first written. Here are some examples of Lift abstractions used by Automata Tutor.

Figure 2.0: “Boot.scala” (Automata Tutor sitemap)

Lift keeps configuration parameters in a file called “Boot” which executes when a Lift application first starts. This includes a sitemap which defines navigation and access control for the entire site (Pollak, 2011).

In Lift, snippets are Scala functions used to generate dynamic content from static HTML. They do this by taking an input NodeSeq (collection of XML/HTML nodes) and outputting a NodeSeq. During this process there can be side effects such as database transactions. This separates HTML design from business logic because program execution is abstracted away from HTML instead of embedded within it (Pollak, 2011).

```
<lift:embed what="/dfa-minimisation-problem/applet-set"> </lift:embed>
```

Figure 2.1: Example of HTML Lift instruction

In Automata Tutor the web app contains various ways to invoke Scala. One way is using “lift:” HTML tags which can be used to instruct the Lift Framework to invoke Scala methods. For example, <lift:something/> tells Lift to invoke the “something” Scala method. In the example above, “embed” is a piece of Scala that comes with the Lift Framework and the “what” attribute is an input parameter to the embed method (Lift Framework, 2021). In this case it specifies the HTML file to load at this position in the current file. The “embed” function is used to stitch together HTML templates.

Scala can also be invoked via “lift:” HTML attributes. An example of this can be found in the “default” HTML file which contains ubiquitous aspects of Automata Tutor such as the header, footer, and navigation bar: . This piece of HTML causes the navigation menu (the site map) to be displayed.

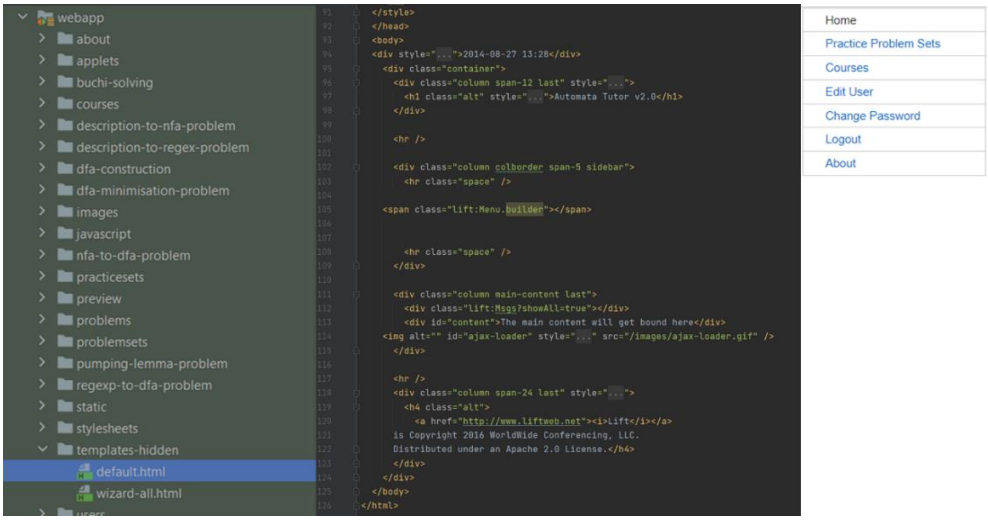


Figure 2.2: “default.html”

Finally, the example below shows an HTML tag that invokes a Scala method from Problem Snippet.

```
<lift:Problems.rendercreate />
```

Figure 2.3: Example of HTML Lift instruction

Scala

Scala is a functional language that grew out of Java and supports object-oriented programming. Functional programming languages only use functions to write programs and often involve recursion. Functional programs are generally useful for solving problems in effective and simple ways as well as improving modularity (freeCodeCamp, 2020). Automata Tutor uses it for business logic.

Scala Build Tool

"Build tools automate the process of building an executable" (Kandhway & Theraja, 2019) as well as managing dependencies. Scala build tool (SBT) is a commonly used build tool for Scala.

C# and ASP.NET Web API

"C# is a modern, object oriented, and type safe programming language" (Wagner, 2021). C# is part of the .NET ecosystem. ASP.NET is a web framework for building web sites using .Net (Carroll, 2015). Automata Tutor's backend grading engine is written in C# and uses the ASP.NET framework to implement a website which responds to requests from the Scala middle-tier part of the front-end project.

HTML, CSS, and JavaScript

Automata Tutor's User Interface is written in HTML, CSS, and JavaScript. HTML describes the basic structure of a webpage and is extended with CSS and JavaScript. CSS is used for formatting and layout. JavaScript makes web pages interactive and dynamic. It is executable code that can be run directly by the browser (Cox, 2020).

H2 Database

H2 is a lightweight, fast database engine that developers can run locally on their computers (Mueller, 2006). H2 comes with a simple, browser-based console application.

H2 was sufficient for the purposes of this project. Its main advantage is that it requires almost no set-up to install and start using. If the project was deployed to servers, then a more powerful database product would be used, like PostgreSQL. It would be easy to make the project use PostgreSQL, by changing the database settings in the "props" configuration file (see Configuration section) but installing and configuring PostgreSQL itself would require learning about another new technology.

Integrated Development Environments

IntelliJ was used for the frontend since a large portion of it was written in Scala. The Scala website also suggests the use of IntelliJ (Martin Odersky, 2021) as it has a convenient Scala plugin.

Visual Studio was used for the backend grading engine which was written in C#. This is simply because it is commonly used for development of C# and ASP.NET applications and was straight forward to setup.

Ethical Considerations

For this project there was no testing with human participants (see Testing Section). Therefore, it was unnecessary to be concerned with the ethics of testing. Furthermore, protecting participant data was unnecessary as there was none.

All information in this document is either original or was sourced from the public domain in the form of books, articles, websites etc. and referenced. There was no test data such as neural network training data etc. as it is not relevant to this project. The Automata Tutor code was accessed from the GitHub repository where it is open source (available here: https://github.com/AutomataTutor). No laws have been broken to access it.

Design

Automata Tutor from a User's Point of View

There are two main types of users. Admins can create a course for students. They can also create Automata exercises for students and add them to a problem set. They can then pose these problem sets for students to complete in a course they have setup.

Automata Tutor makes creating and editing exercises easy. When creating new exercises, there is no need to provide model solutions because Automata Tutor's grading engine does this automatically.

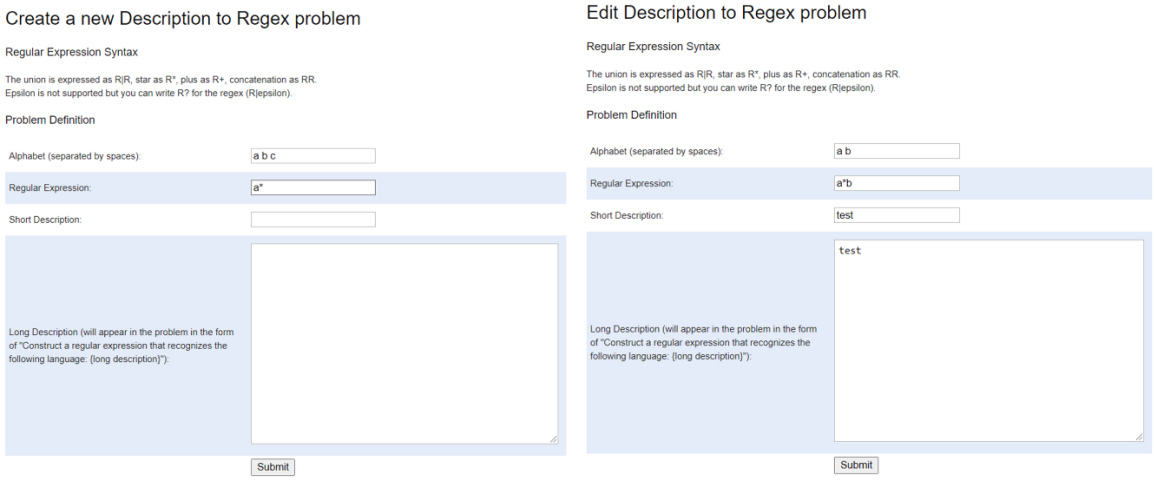


Figure 4.0: Creating and editing a description to Regular Expression problem

Students can enrol on their teacher's course with a course ID and password. From here they can access posed problem sets and complete exercises. The grading engine provides grades and feedback.

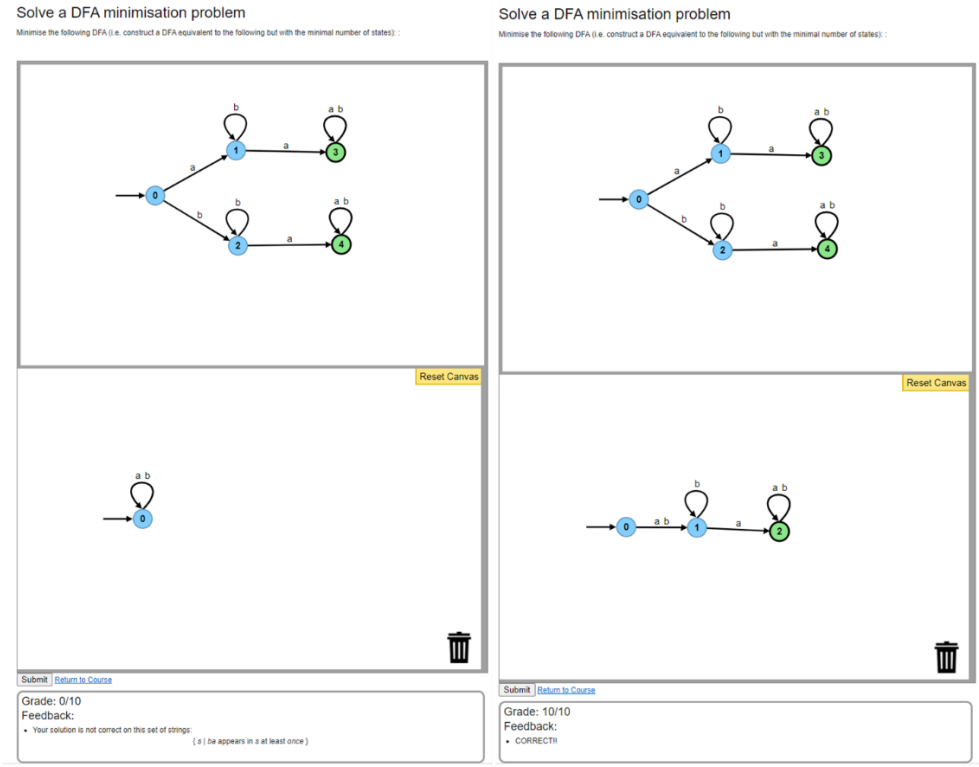


Figure 4.1: Solving a DFA Minimisation problem

Data Model

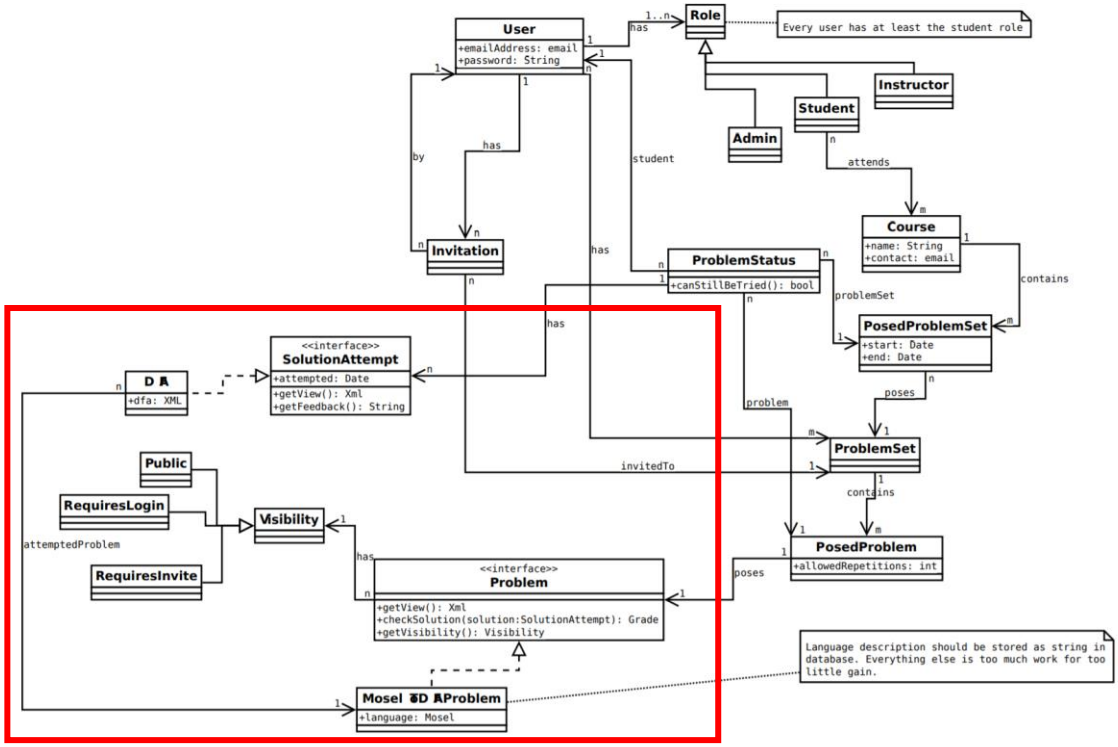


Figure 4.2: Data Model (Weinert & D'Antoni, 2019)

This is the data model. It is part of the documentation on GitHub made by the creators. It matches the description given from the user's point of view. For example, users have an abstract role type with concrete implementations such as admin and student etc.

For this project, it was only necessary to work on the part highlighted in the red box from Figure 4.2. The rest was ignored. This is because it already has a full implementation and is designed to work with any problem type you try to add. This is part of the creators' aims to make adding new problem types as easy as possible.

The Problem classes, model problem types. The Lift object relational mapper (ORM) is used to persist and read objects to/from the data store.

The SolutionAttempt classes model student answers to problems. They contain the grade and time taken etc.

System Architecture and Design

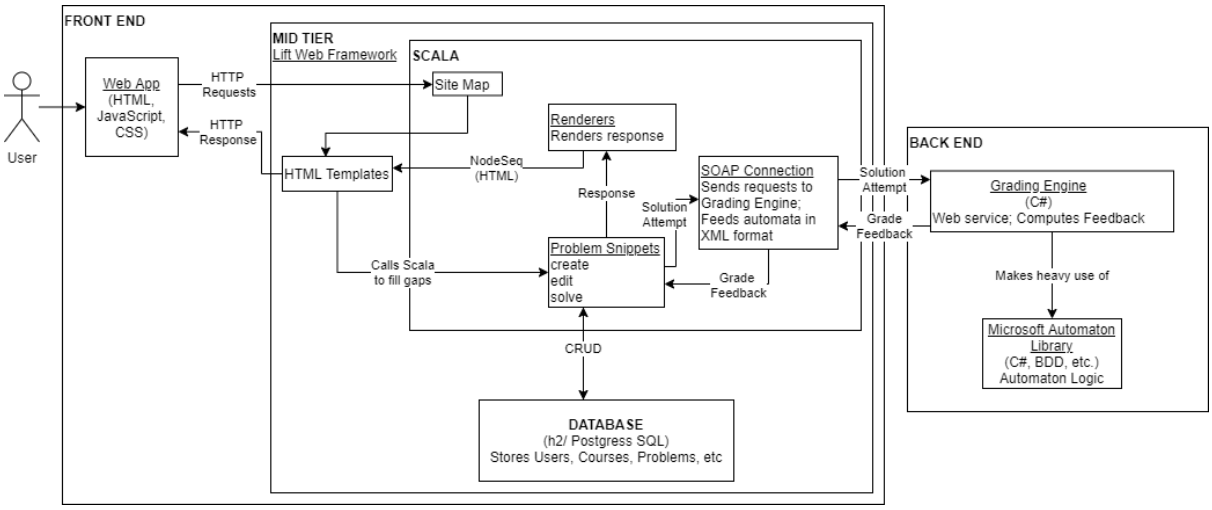


Figure 4.3: System Architecture and Design Diagram

Figure 4.3 illustrates the system architecture. A large proportion of the time spent working on this project was devoted to acquiring the knowledge to produce this diagram.

The front end is split into two parts. The web app and the middle tier. The web app is the interface between the system and the user. It is written in JavaScript and HTML, and runs on the user's client, in the browser. When a user interacts with the web app on their client, an HTTP request is made to the middle-tier to perform a state transition.

When the web app makes an HTTP request to the middle tier, the Lift Web Framework checks a sitemap to retrieve an HTML template. The template has "gaps" that need to be filled before the response can be sent back to the client. Lift runs the appropriate "snippet" (Scala function) to dynamically generate xml which replaces the "content" tags in the template.

In the Lift Web Framework, snippet means a Scala function that takes a sequence of HTML elements as input (a NodeSeq) and produces a new sequence to replace it in the response to the client. Hence, for example, a simple "place holder" element might be replaced by a complex table of data that has been read from a database (Pollak, 2011).

Business logic for the front-end site is implemented in Scala. This includes:

- The sitemap.
- Data persistence and retrieval, to/from a database.
- Interacting with the grading engine (the back end), which scores students' answers to problems.
- Generating the dynamic content of web pages.

The database stores users, courses, problem sets, problems, grades etc. A Scala problem snippet loads objects from the database and loads HTML templates for various functionalities such as creating problems or solving them. In this version of Automata Tutor, the creators were in the process of separating the responsibility of rendering a page and querying the database. However, this process was incomplete, so in the current implementation of these tasks are mostly in "Snippet".

When a student is solving a problem and presses submit, the web app needs to respond with feedback. The back-end grading engine, written in C#, is responsible for grading a student's solution. It is a separate application from the front end and runs independently. The middle tier in the front end, has a "SOAPConnection" Scala class which is called by a "snippet" to communicate with the grading engine. It makes requests and sends student's attempt solutions as well as their question prompts.

The "Service1" class in the grading engine handles requests to compute feedback for all problem types. The grading engine makes heavy use of the Microsoft Automata Library. This library performs useful operations on automata. For computational convenience, xml representations of automata are converted into Automata objects which contain Binary Decision Diagram representations. "Service1" eventually calls a "getGrade" method from a "Grading" class to make high level comparisons via the Microsoft library and return a grade with feedback.

This feedback is returned to the "SOAPConnection" which is loaded and rendered by "Snippet" and inserted into the html template where the chain of requests started and handed back to the user's client. (Alexander Weinert, 2019)

This is a high-level summary of the system. The Automata Tutor Read Me also provides similar documentation with added detail over aspects of the front end. This was helpful to obtain a broad idea of how the system worked but frankly it was brief. More documentation certainly would have been helpful. It was also misleading by suggesting implementation of a problem snippet would be the most challenging part of adding a new problem type. Though this was indeed a crucial part of the project, the grading engine which itself had extremely brief documentation was equally important. Understanding the grading engine and Microsoft's Automata library and how it could be used was essential to adding a new problem type. Acquiring this knowledge mostly had to be done by deciphering code and through "Software Archaeology" since the grading engine in this version of Automata Tutor used an ancient version of the Microsoft Automata library.

Microsoft Automata Library

The Microsoft Automata Library is a toolkit for symbolic Automata. In the context of Automata Tutor, it is used in the feedback engine to operate on Automata. The main part of the library is the Automaton class. It is defined as a symbolic Automaton over some representation that supports Boolean operations (Veanes & Bjørner, 2012).

It is possible to convert Automata into truth tables which is useful for representing them on modern computers. However, this is computationally expensive and consumes a lot of memory since there needs to be a row for every permutation of states to indicate if there is a transition between them.

A more standard representation of Automata on computers is with binary decision trees (BDD). This is an expressive and memory efficient method because BDDs can be compressed and simplified whilst still retaining the same amount of information. It also makes it possible to write algorithms for Automata operations with smaller time complexities (Srivathsan, 2015). This is the representation Automaton objects in Automata Tutor use.

Conversion of Automata to BDDs, the properties of BDDs, reducing them and, their use in the Automata library, are fascinating academic studies. However, it goes beyond the scope of this project since a general awareness of what they are and how and where they are used is sufficient.

Implementation

Building and Running Automata Tutor

The Backend is written in C# .Net. It is straight forward to download the community edition of Visual Studio and compile and run this.

The Frontend is trickier to get running. The community edition of IntelliJ is a good IDE for Scala as it has a Scala plugin and support for SBT. The project must be configured to use Java version 1.8. The project will not run until the "props" file contains valid database and grader (Backend URL) settings. This is explained later.

To run the frontend in IntelliJ, make sure the backend is already running. Open the sbt console and execute the following commands:

```
compile

container:start
```

Note that this is not sufficient for debugging. A later section describes how to debug the middle tier. To stop the application run the following command:

```
Container:stop
```

Debugging Automata Tutor

To debug the application there are three different places to be looking. The first is the chrome dev tools which are well documented. This allows the content of problems and student answers to be inspected as they are sent to the Scala middle tier (Google, 2021).

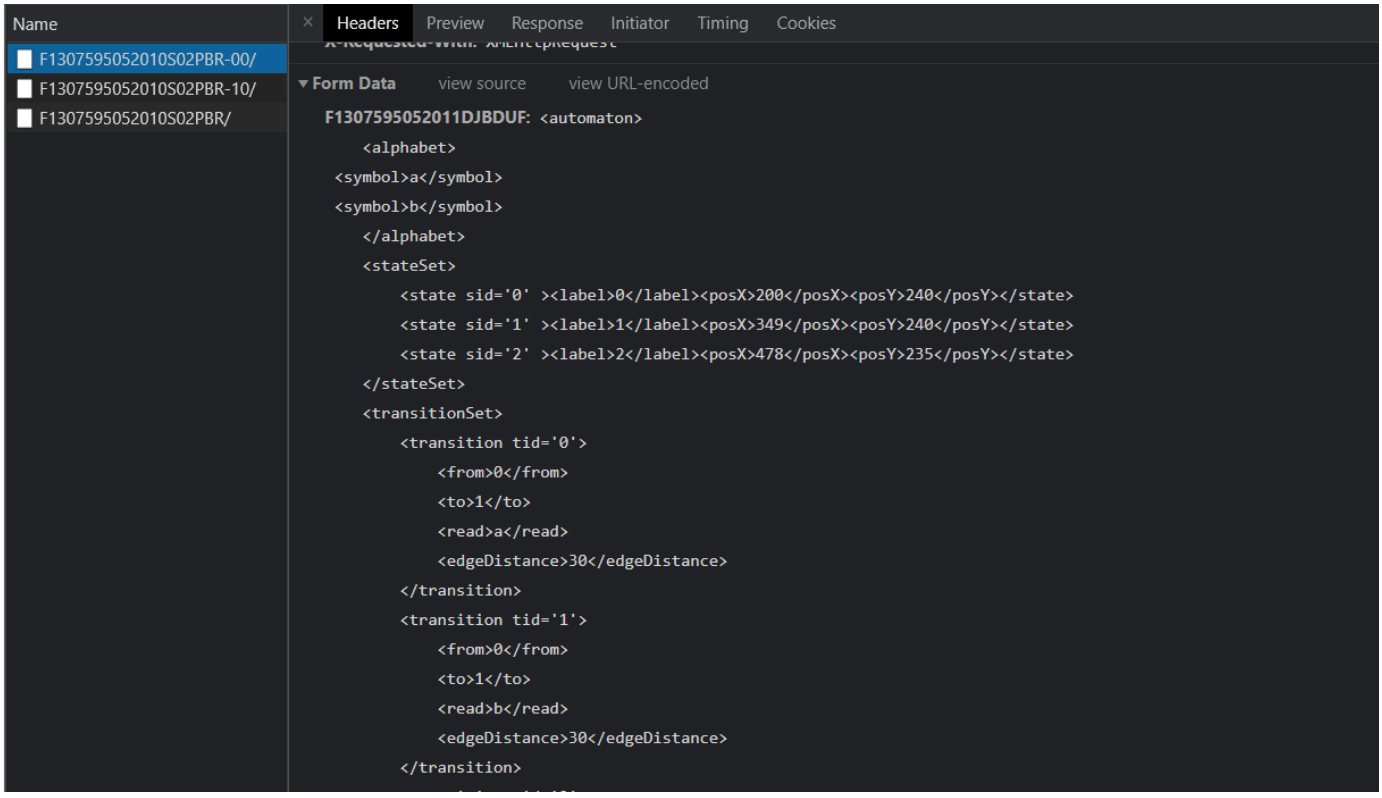


Figure 5.0: XML representation of a student's attempt solution as revealed by Chrome Dev Tools

Next in terms of straight forwardness is the .Net backend which can be launched from Visual Studio simply by pressing the debug button (or pressing F5). This brings up the backend web service in a browser.

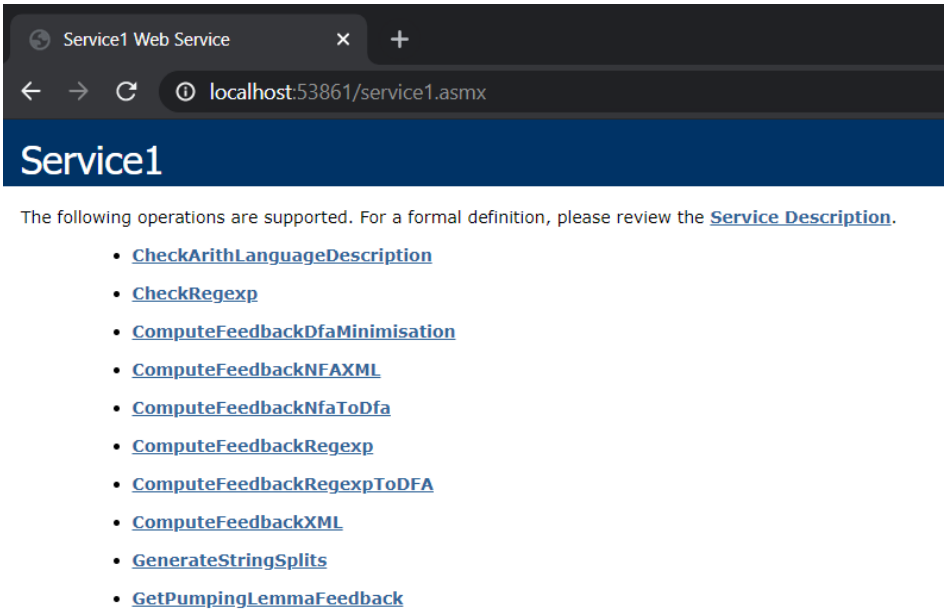


Figure 5.1: Grading engine web service

Break points can be set in the .Net code to inspect the data that is being sent from the frontend and the responses that are being generated. Unfortunately, it proved beyond this author's abilities to figure out how to step into the Microsoft Automata library, which is heavily depended on in the grading engine, especially because it was unclear which version of the source code Automata Tutor was based on.

The trickiest thing to debug was the Scala middle tier. The following Stack Overflow article provided the "magic" command line (Stack Overflow, 2011).

```
java -Xmx512M -Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005 -jar
.\sbt-launch.jar
```

Available at:

https://stackoverflow.com/questions/8621542/how-do-i-debug-lift-applications-in-eclipse

Once the middle tier had been launched in this way it could then be connected to from IntelliJ and breakpoints could be set.

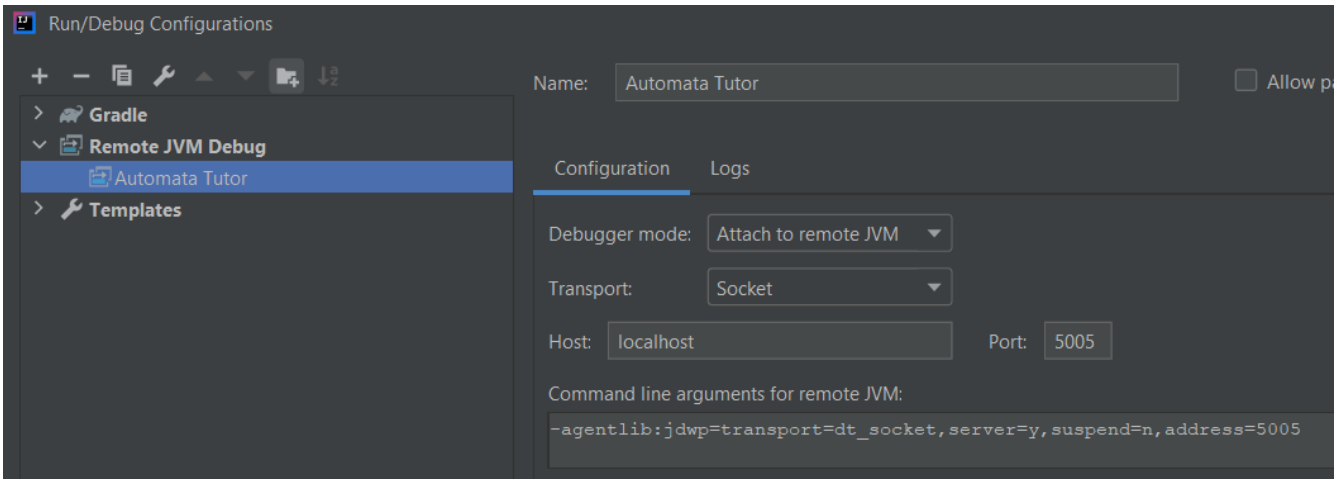


Figure 5.2: IntelliJ debug configurations

Configuration

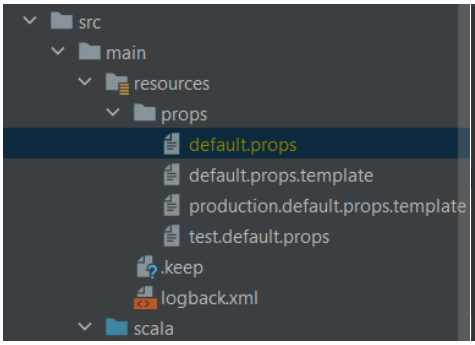


Figure 5.3: “Props” file directory

```
db.driver=org.h2.Driver
#db.url=jdbc:h2:mem:
db.url=jdbc:h2:file:./dev.db
db.user=
db.password=


grader.url=http://localhost:53861/Service1.asmx
grader.methodnamespace=http://automatagrader.com

admin.email=admin@somewhere.com
admin.password=test1
admin.firstname=Donald
admin.lastname=Knuth
```

Figure 5.4: Contents of “Props” for this project

Under the Lift Framework, there is a “props” file which contains important information used by the system. The initial admin user, connection to the Grading Engine, and connection to the Database is configured here. In Automata Tutor the “props” file is found in the following directory: “src/main/resources/props/default.props” (Weinert & D'Antoni, 2019). Automata Tutor comes with a template which can be adapted for development.

An initial admin account needs to be specified to login and use Automata Tutor on a local machine. This account can be used to make other admins, although for development this was not necessary.

Accessing the Database

To connect to the database, the system needs to know what database technology to use, where the database can be found, and what username and password to use to login to it. This is configured in “default.props”. For local debugging of Automata Tutor, the h2 database engine can be used. The database can be persisted to a local file by using the “file” option in the db.url setting (h2, 2021):

```
db.url=jdbc:h2:file:./dev.db
```

With this in place it was possible to inspect the database which was useful for reverse engineering the application. To make it easy to view the database, the h2 jar file was copied from where it is awkward to reference in the Java library cache, to the root of the application directory, where it is always easy to find. With this in place, the following command can be run in a command prompt:

```
java -cp h2-1.4.182.jar org.h2.tools.Server
```

This brings up a simple database browsing app in a web browser. Leaving the “db.user” and “db.password” fields blank in “default.props” made it easier to work with the local database because credentials did not need to be remembered or entered as shown in the screenshot below.

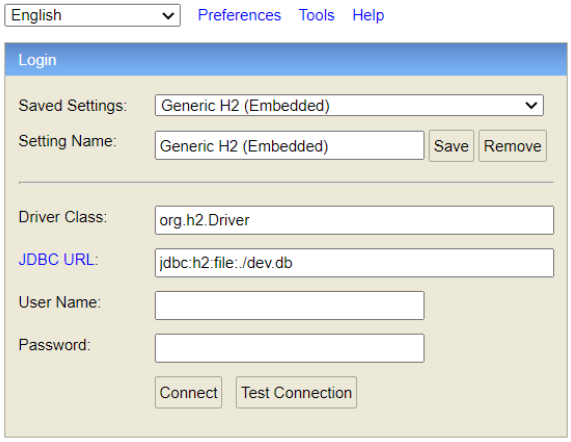


Figure 5.5: h2 database browsing app

After connecting it is possible to query and change the database as shown in the screenshot below. This proved to be important because it was necessary to manually mark test email addresses as valid, since they were not real. This was required because a user cannot logon to the application until their email address has been marked as valid in the database.

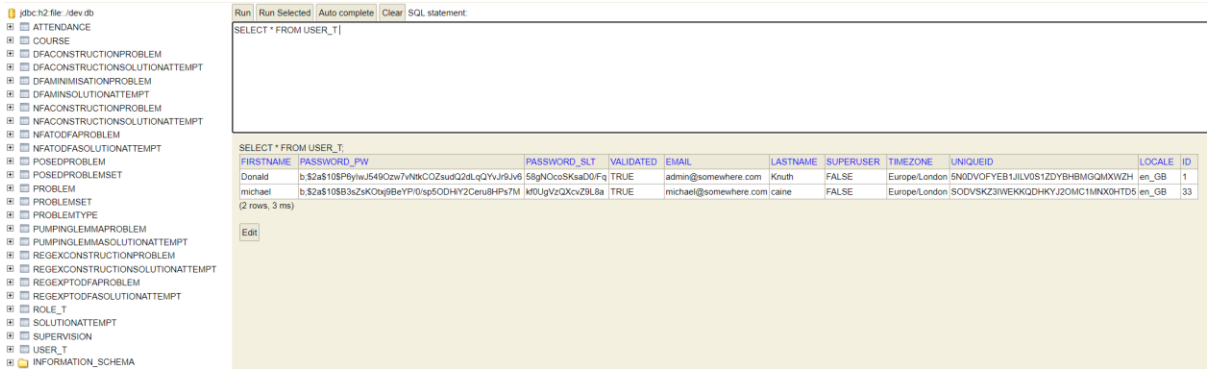


Figure 5.6: h2 database browsing app

Adding a New Problem Type to Automata Tutor

Adding a new problem type requires making changes to all tiers of the application:

- Frontend: how the problems of this type are presented to the tutor setting questions and the students solving them (two different views).
- Middle tier: How the problems are modelled and persisted.
- Backend: How student solution attempts are graded.

Adding a New Problem Type to the Frontend and Middle Tier

The Read-Me file provides guidance on extending the Frontend project with new problem types as shown below.

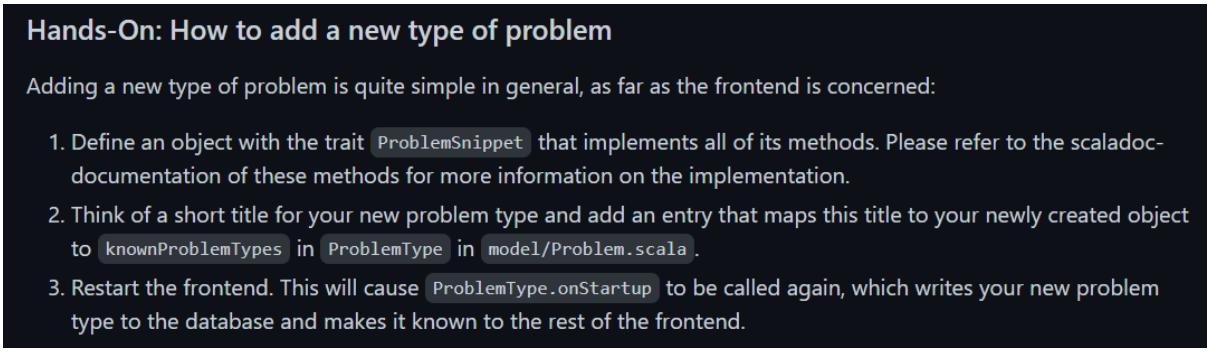


Figure 5.7: “How to Add a New Type of Problem” from the GitHub Read-Me (Weinert & D'Antoni, 2019)

Adding a New Problem Snippet

The first thing to add is a new Problem Snippet. Problem Snippet classes are the heart of the frontend (middle tier). They are responsible for interacting with the database, generating HTML, XML and JavaScript for the webapp, as well as communicating with the grading engine via the SOAP Connection object.

All Problem Snippets contain a “create”, “edit”, “solve”, and “onDelete” function. The easiest approach for implementation was to determine the most similar existing problem and clone it to use as a template. Then, to begin refactoring it for a new problem type.

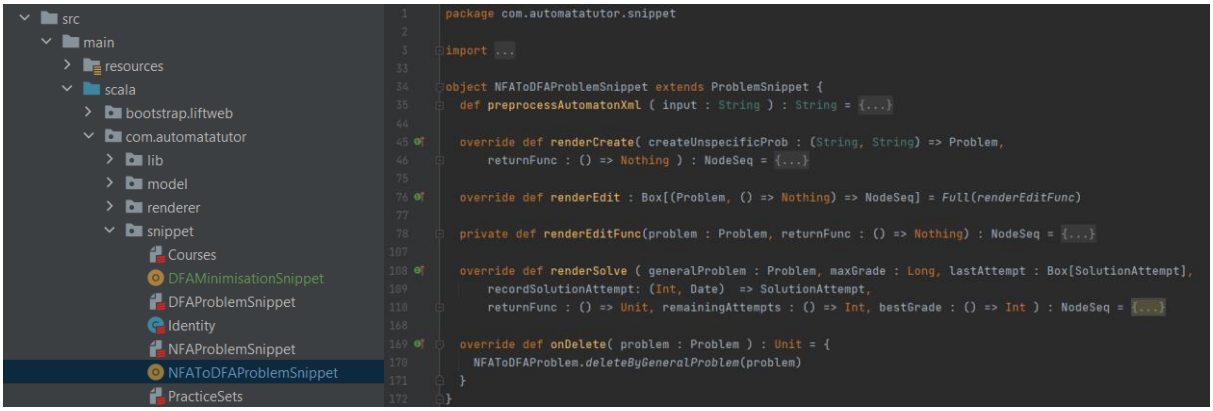


Figure 5.8: Example Automata Tutor Problem Snippet “NFAToDFAProblemSnippet.scala”

The “NFA to DFA” problem type presents a student with an NFA which they must convert to a DFA. This was copied and adapted for “DFA Minimisation”. The NFA is replaced with a DFA which the student must minimise.

“Regular Expression to DFA” was implemented by adapting “create” and “edit” Scala functions from “English to Regular Expression”. The “solve” function was adapted from “English to DFA”.

Recognising the New Problem Type

“Problem.scala” defines all problem types known to the application, so it is necessary (Read-Me step 2) to add the new problem type to the classes in “Problem.scala”. This includes, for example, a mapping table of problem type names to problem type Snippets.

To implement the DFA Minimisation and Regular Expression problem types, these were simply added as instructed.

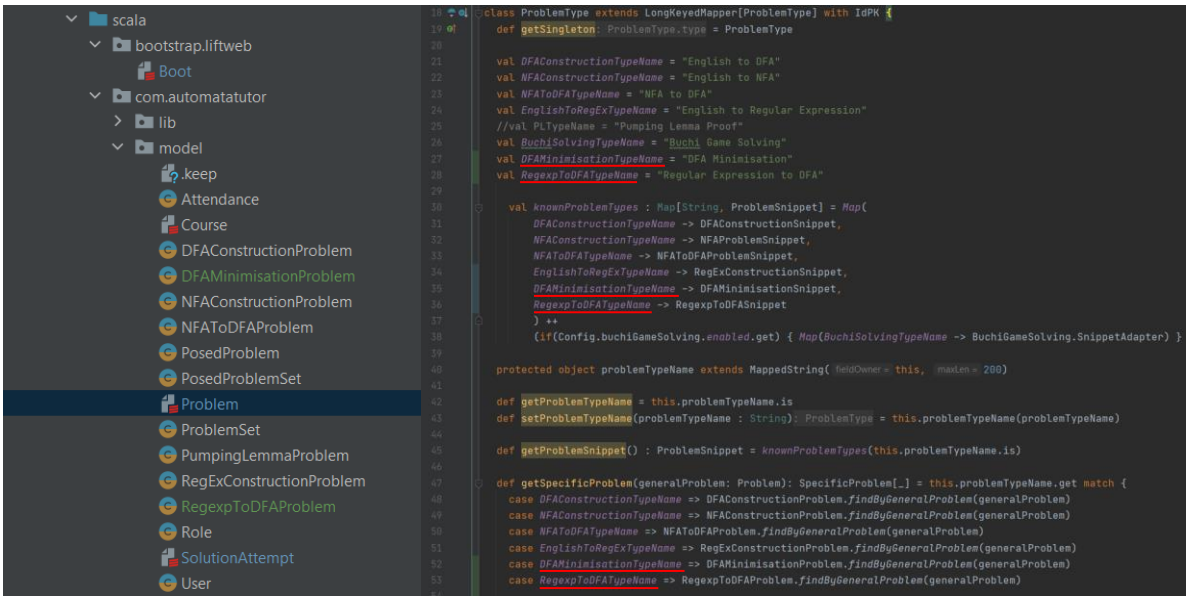


Figure 5.9: “Problem.scala” with the author’s additions.

Recognition of the new problem types can be checked by running the frontend in the local host and signing in as admin to create a new problem. Here, the new problem types appeared.

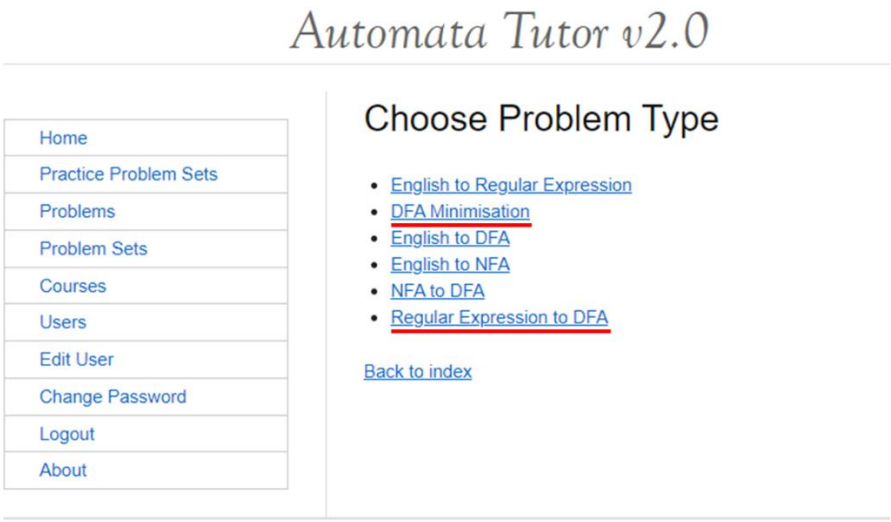


Figure 5.10: Checking new problem types are recognised by the system.

Implementing Problem and Solution Attempt Types

The “Problem” classes model problem types. As mentioned before, the Lift object relational mapper (ORM) is used to persist and read objects to/from the data store. “Problem” objects represent teacher set exercises and can be loaded when a teacher wants to edit a problem, or a student wants to solve one.

To implement a new problem type it needs a “Problem” class. Once more, the easiest way to implement this is to follow the patterns of existing problem types which for “Problem” classes is mostly the same.

Since the DFA Minimisation Snippet was a variation of the NFA to DFA Snippet, the NFA to DFA “Problem” class was also copied and adapted. Implementation was a case of renaming variables and ensuring things were wired up correctly.

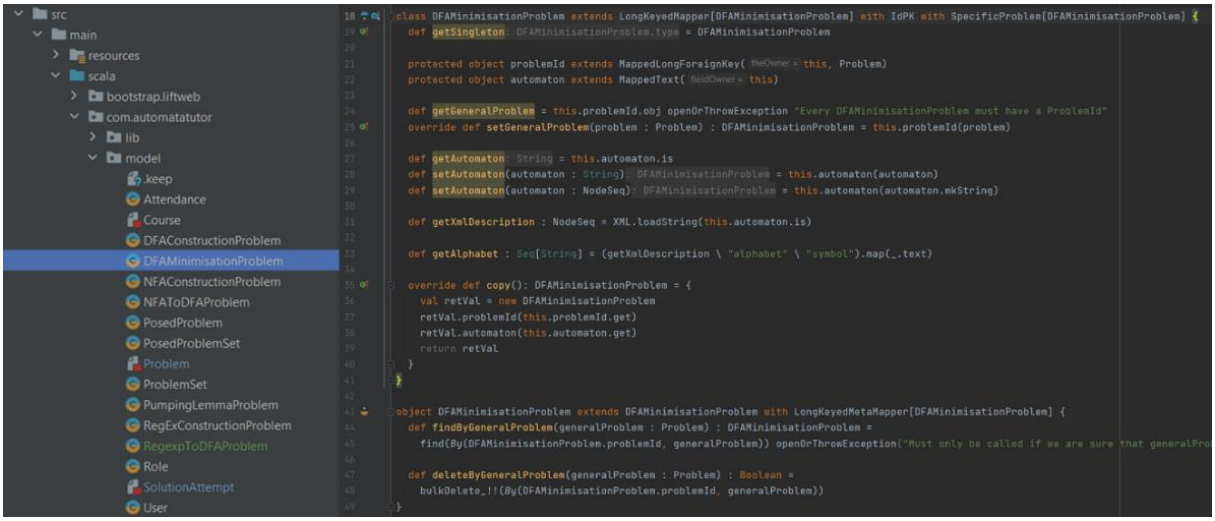


Figure 5.11: “DFAMinimisationProblem” Example of my implementation

The Regular Expression to DFAs “Problem” class is an adaptation of English to Regular Expression’s. Similarly, variables needed to be renamed.

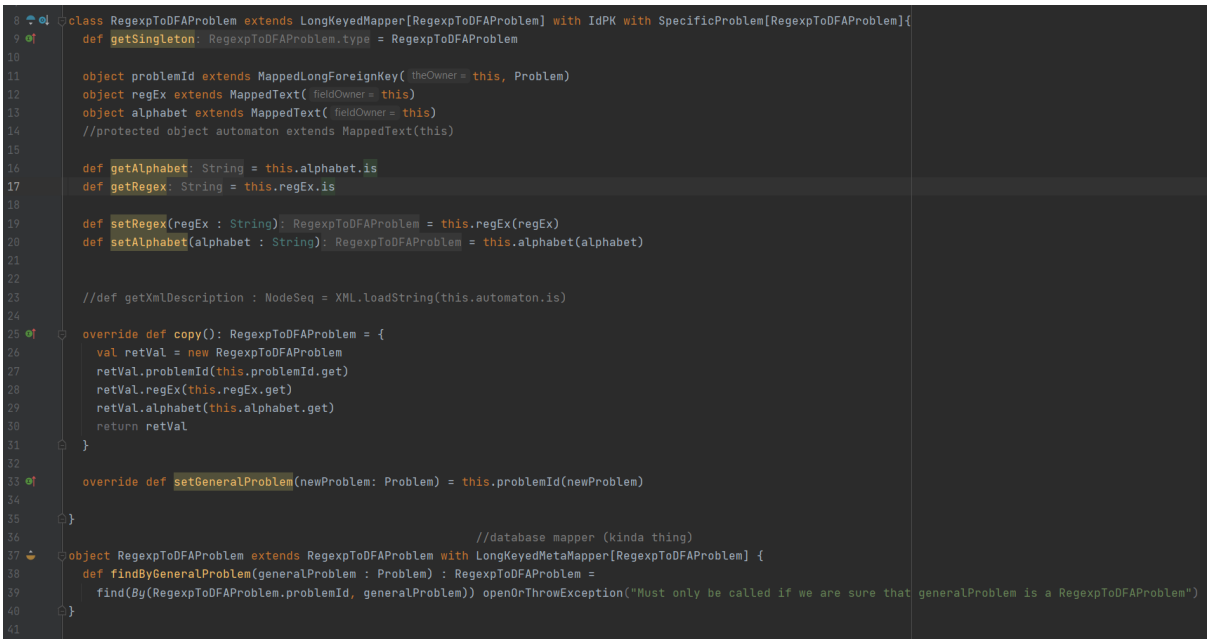


Figure 5.12: “RegexpToDFAProblem” example of implementation

The diagram below shows DFA Minimisation problems stored in the database. These rows in the table can be generated by creating a DFA Minimisation problem when logged into the webapp as an admin. In the Frontend Automata are represented as xml. They are later converted to a more appropriate format for the grading engine.

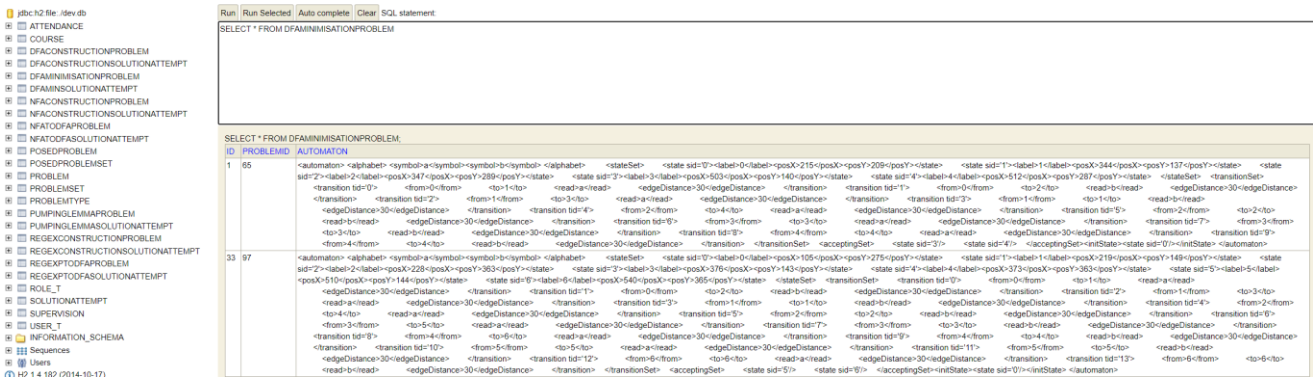


Figure 5.13: DFA Minimisation Problem objects in the database

The “SolutionAttempt” class models the details of student answers to problems. They store the grade, and time etc. There are also “SolutionAttempt” classes specific to problem types e.g. “DFAMinSolutionAttempt”. These contain the actual answers that students gave in their solution attempt. For example, a “DFAMinSolutionAttempt” object contains the Automaton a student inputted.

The pattern for “SolutionAttempt” classes is basic and mostly the same for all problem types. DFA Minimisation and Regular Expression to DFA “SolutionAttempt” classes follow this pattern.

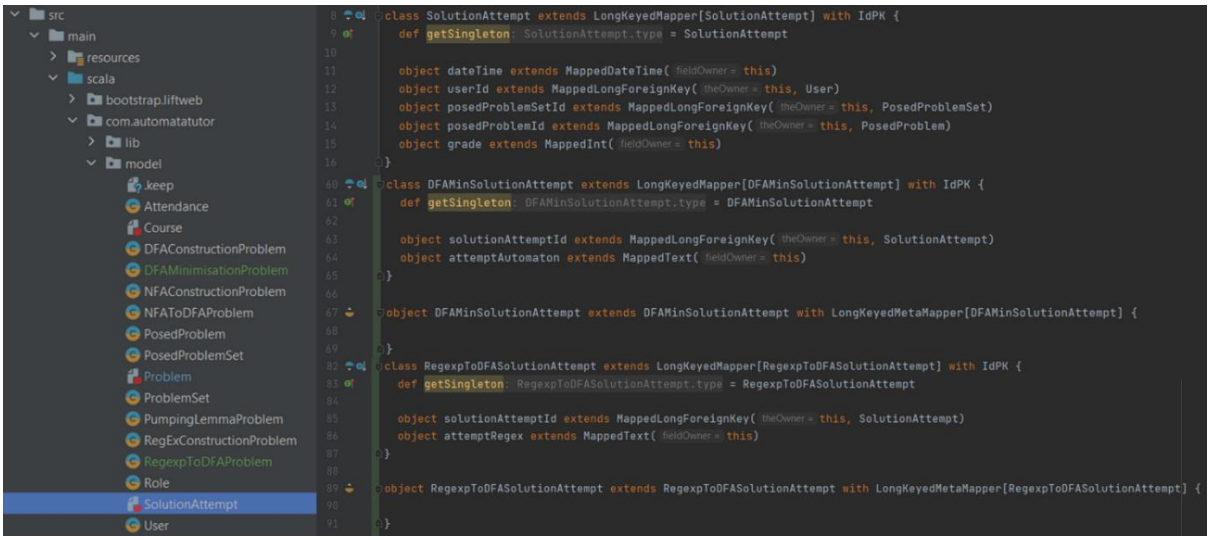


Figure 5.14: Solution Attempt implementation

Solution Attempts can also be viewed in the database by attempting to solve a problem as a student. However, this requires the rest of the system to be implemented. The diagram below is to illustrate the relationship between the general “SolutionAttempt” class with problem specific “SolutionAttempt” classes.

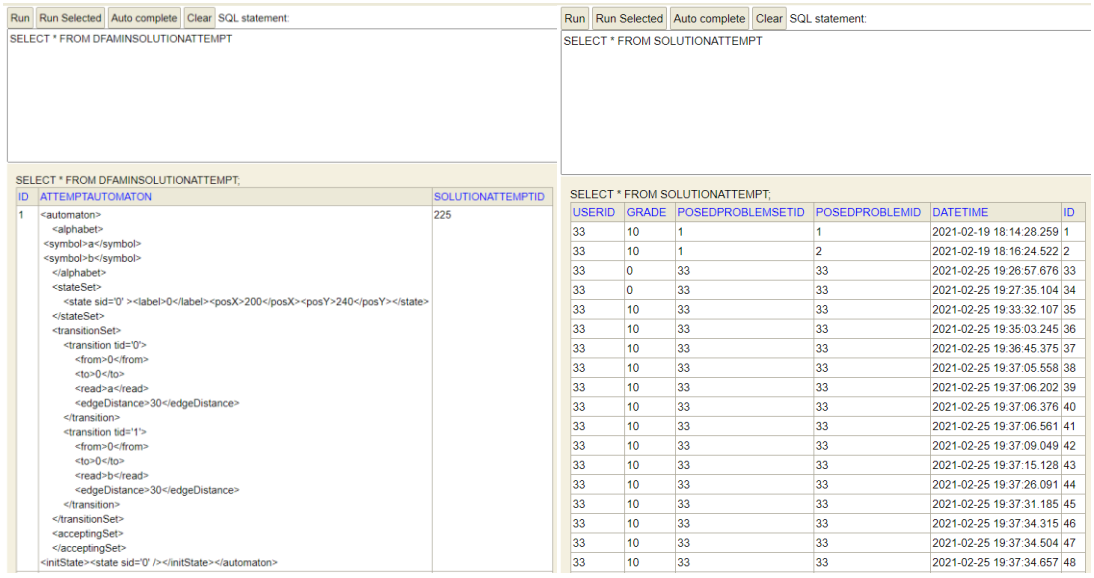


Figure 5.15: DFA Minimisation Solution Attempt in the database

Sitemap

When adding a new problem type, it's "Problem" and "SolutionAttempt" classes need to be added to the sitemap in "Boot.scala". For DFA Minimisation and Regular Expression to DFA, these were simply added to "schemifier.schemify" (underlined in diagram below). This tells the ORM what classes to persist.

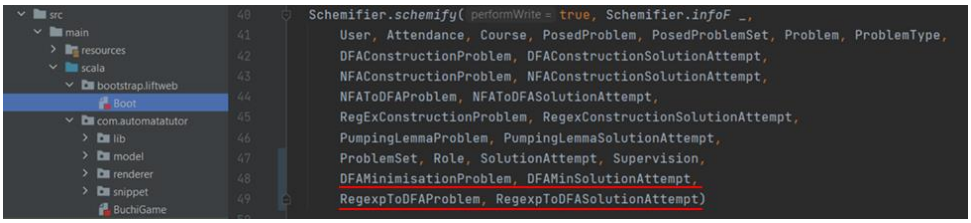


Figure 5.16: "Schemifier.schemify"

HTML Templates

Generally, HTML templates for new problem types need to be made for create, edit, and solve pages. Much like the Snippets, these were adapted from existing implementations. DFA Minimisation adapted these templates from the NFA to DFA templates. Regular Expression to DFA adapted "create" and "edit" templates from English to Regular Expression where as the "solve" template was adapted from DFA construction.

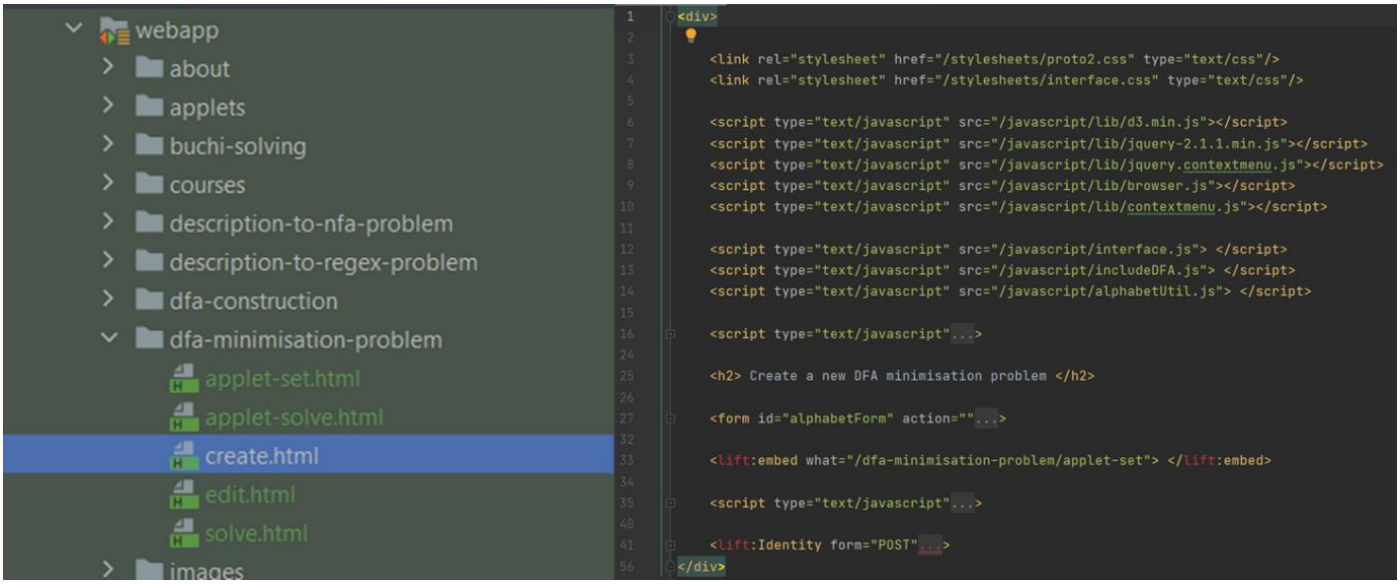


Figure 5.17: DFA Minimisation: create.html

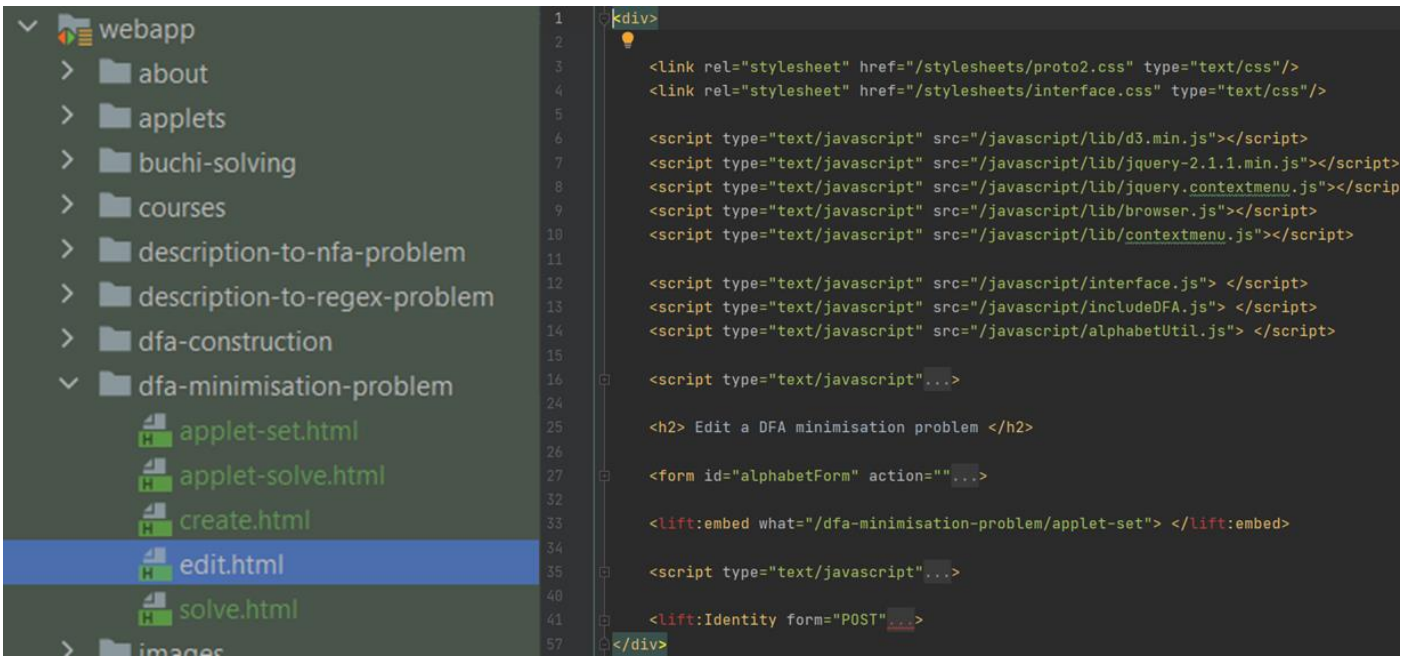


Figure 5.18: DFA Minimisation: edit.html

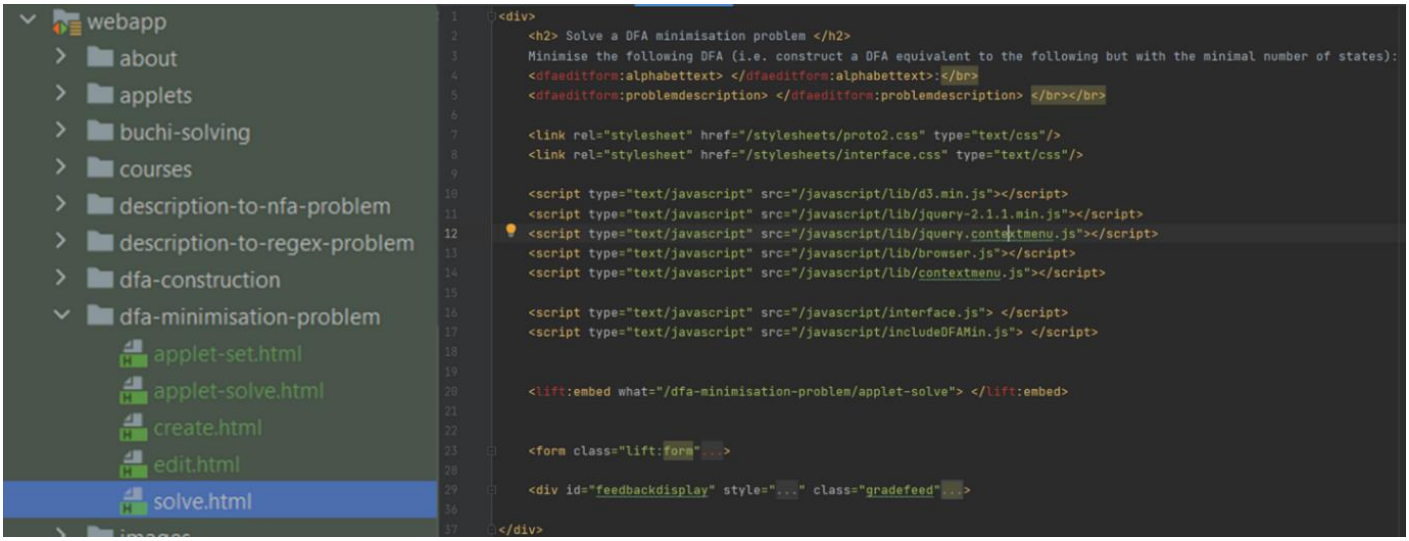


Figure 5.19: DFA Minimisation: solve.html

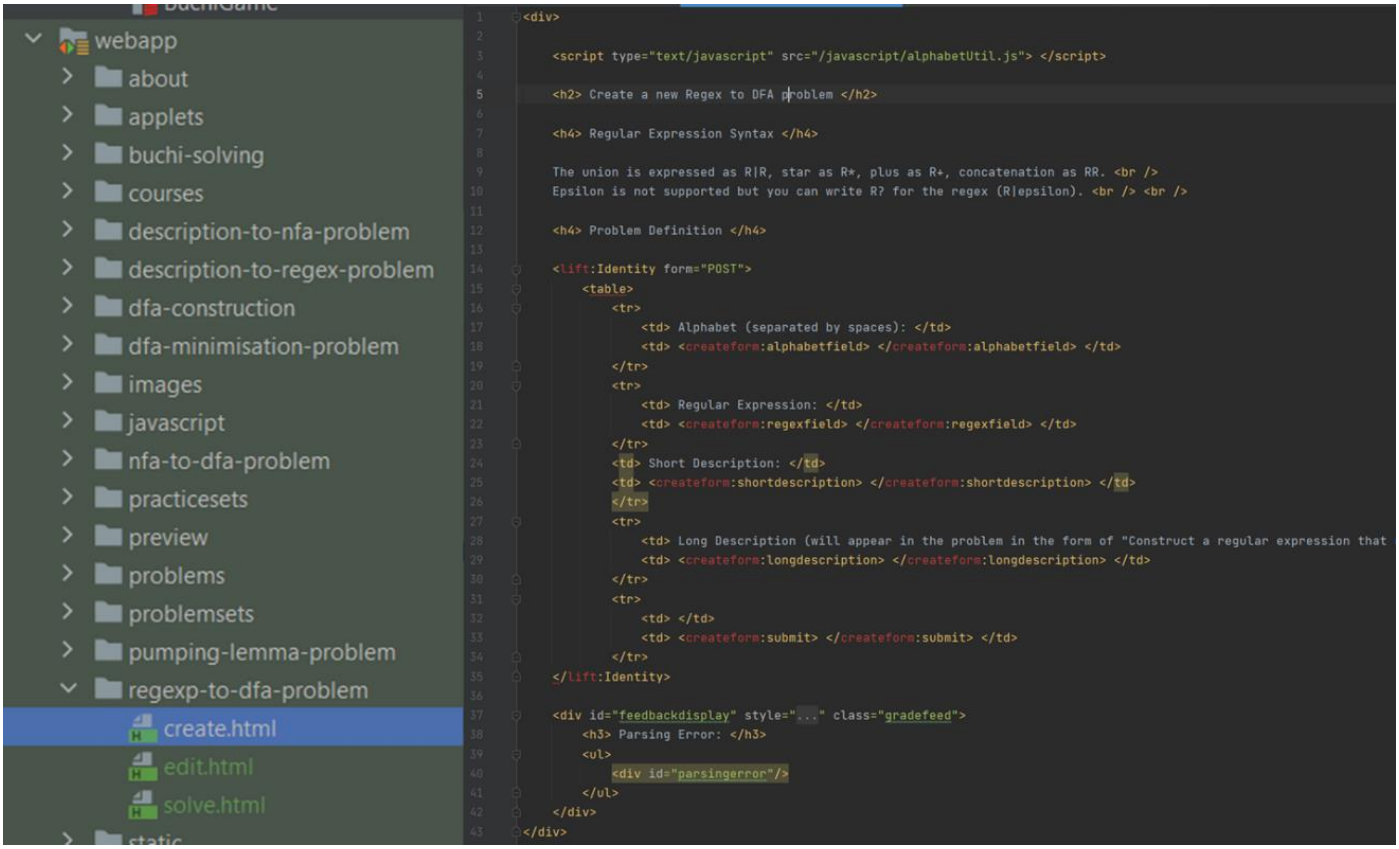


Figure 5.20: Regular Expression to DFA: create.html

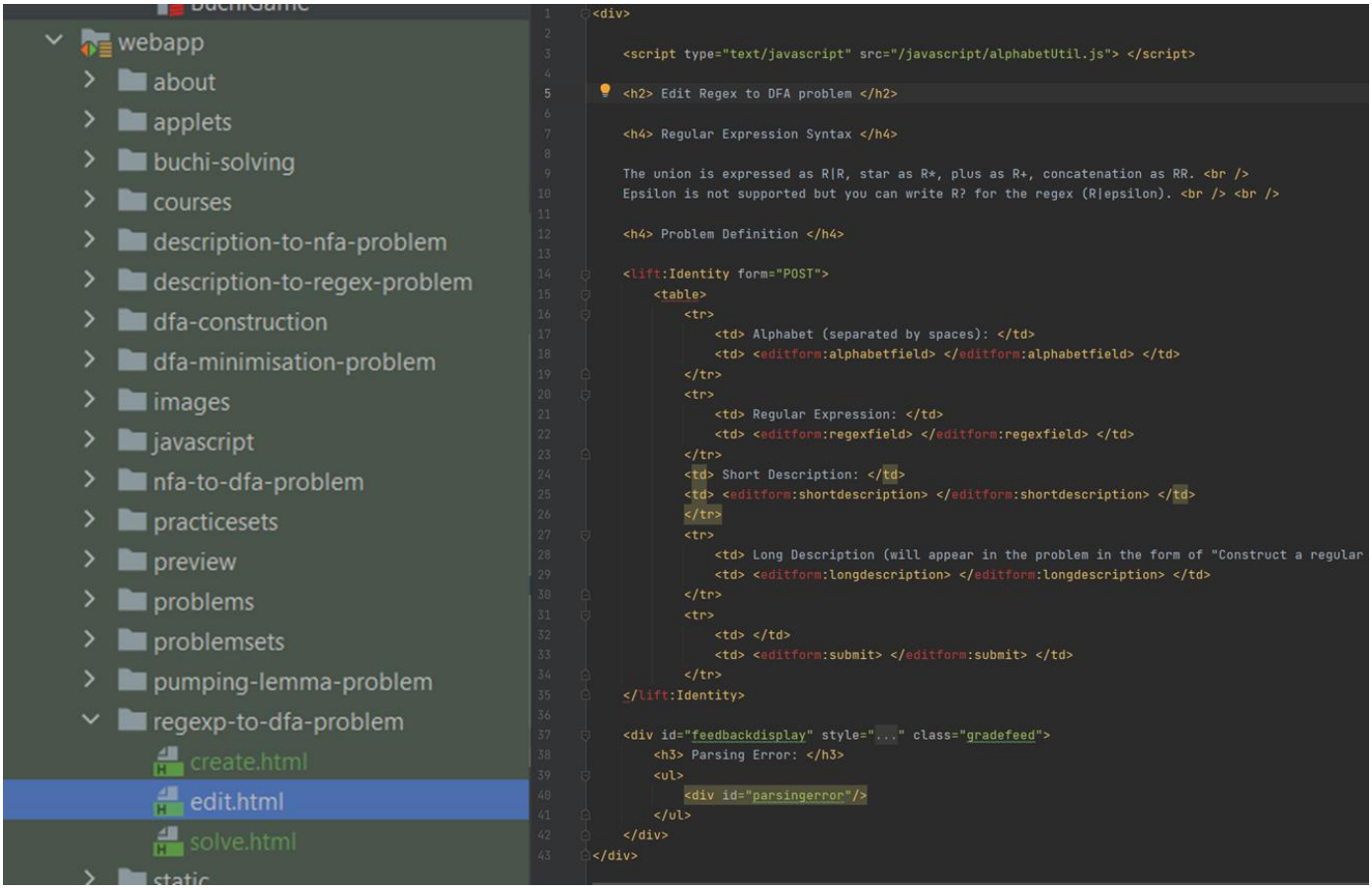


Figure 5.21: Regular Expression to DFA: edit.html

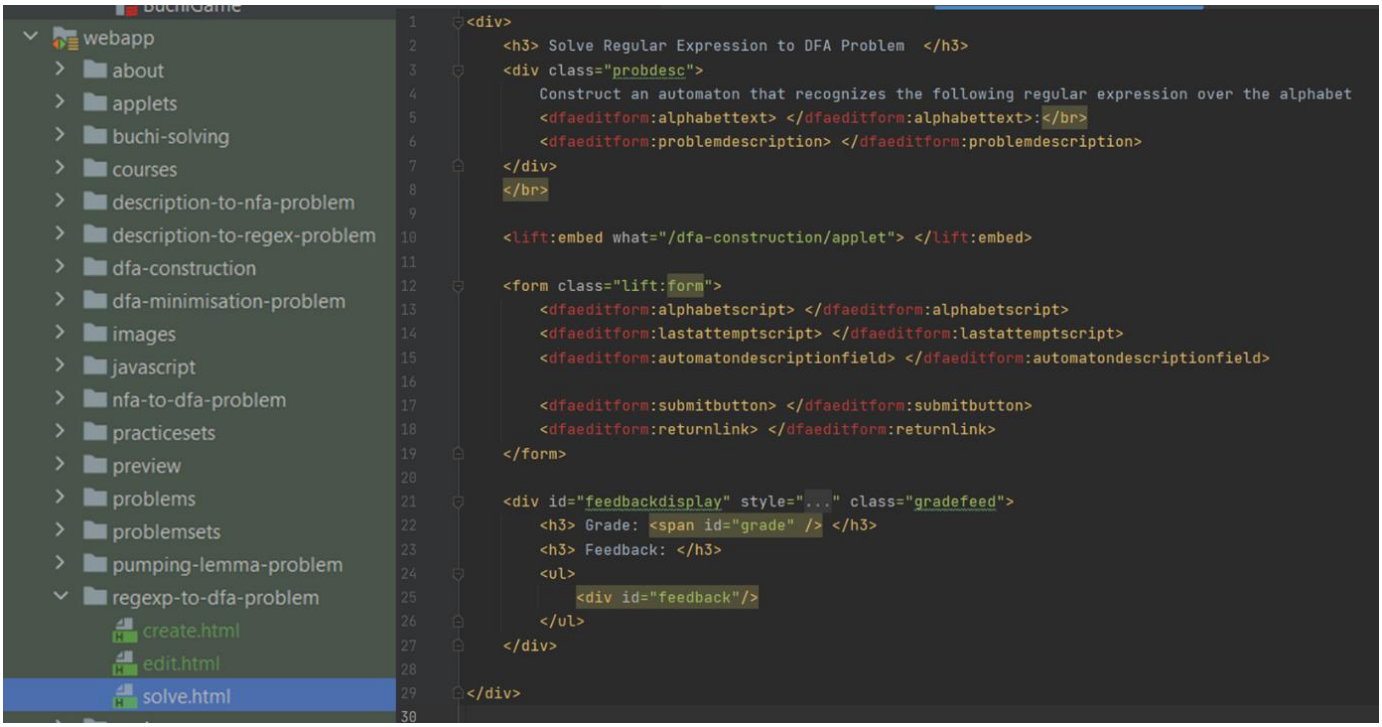


Figure 5.22: Regular Expression to DFA: solve.html

The project came with a nice JavaScript library for drawing Automata and persisting them in an easy to read and understandable xml format. The diagrams are drawn using an HTML 5 canvas (W3 schools, 2021).

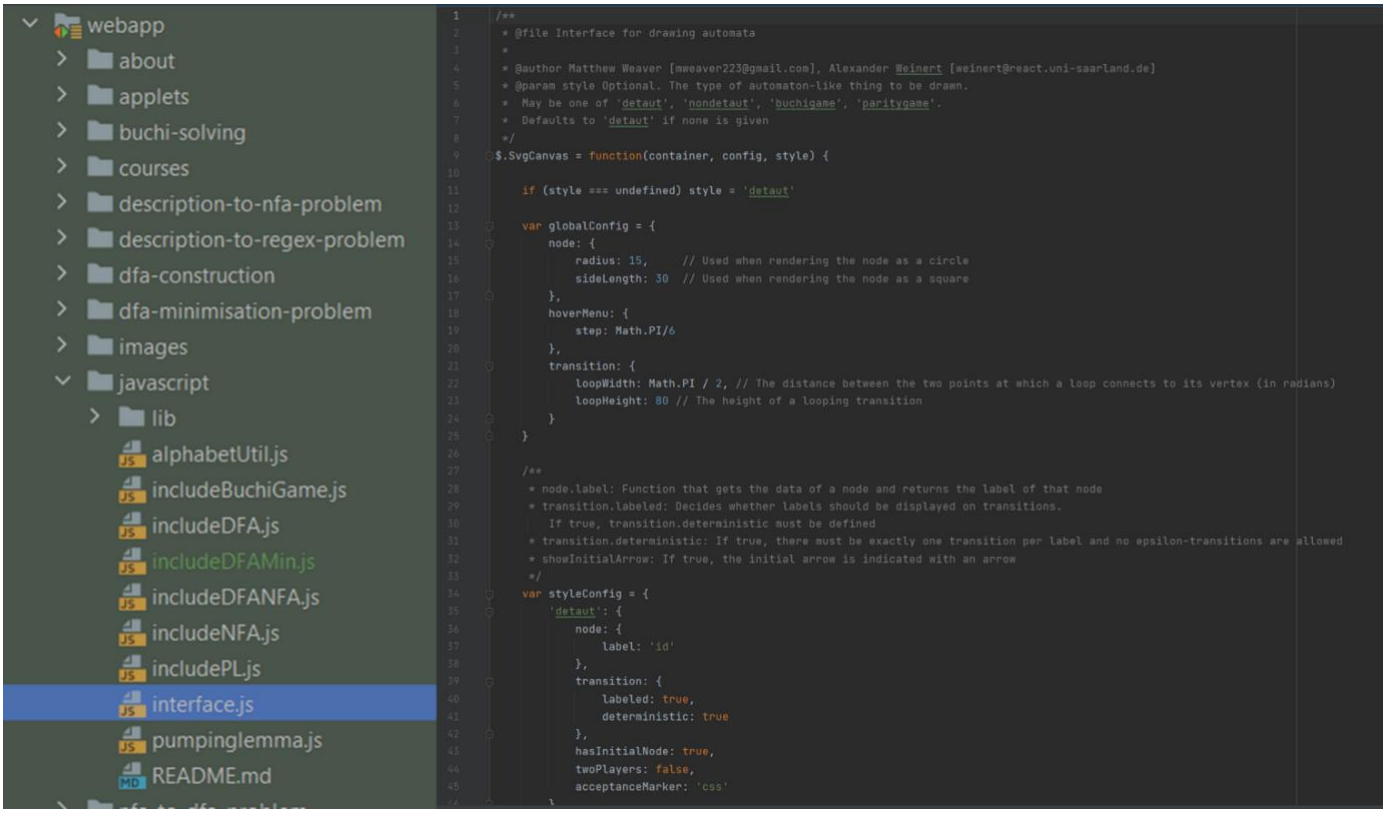


Figure 5.23: Automata Tutor JavaScript library

All HTML templates for problem types reference some form of JavaScript. Implementing JavaScript can be avoided if a new problem type is similar with an existing one because they can reference the same files. This was the case for Regular Expression to DFA. However, particularly unique problem types need custom JavaScript to be written. It was necessary to edit JavaScript for the DFA Minimisation problem in order to present two canvas objects: the non-minimised problem to be solved, and the student's solution attempt. The "detaut" property was set on each Automata canvas, to indicate that both Automata were to be treated as deterministic. This affected how edges could be added to the diagrams.

Figure 5.24: JavaScript for DFA Minimisation

The Snippets then had to be updated to make sure they reference and load the correct HTML templates.

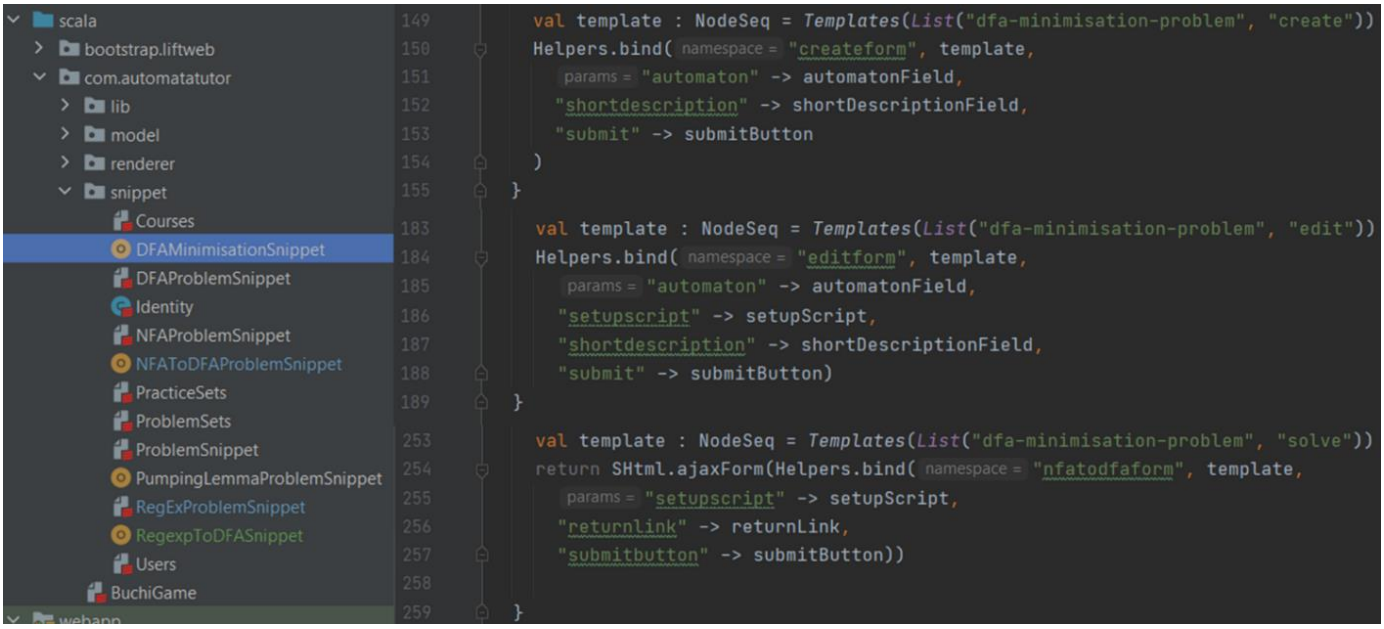


Figure 5.25: References to HTML templates in DFA Minimisation Snippet

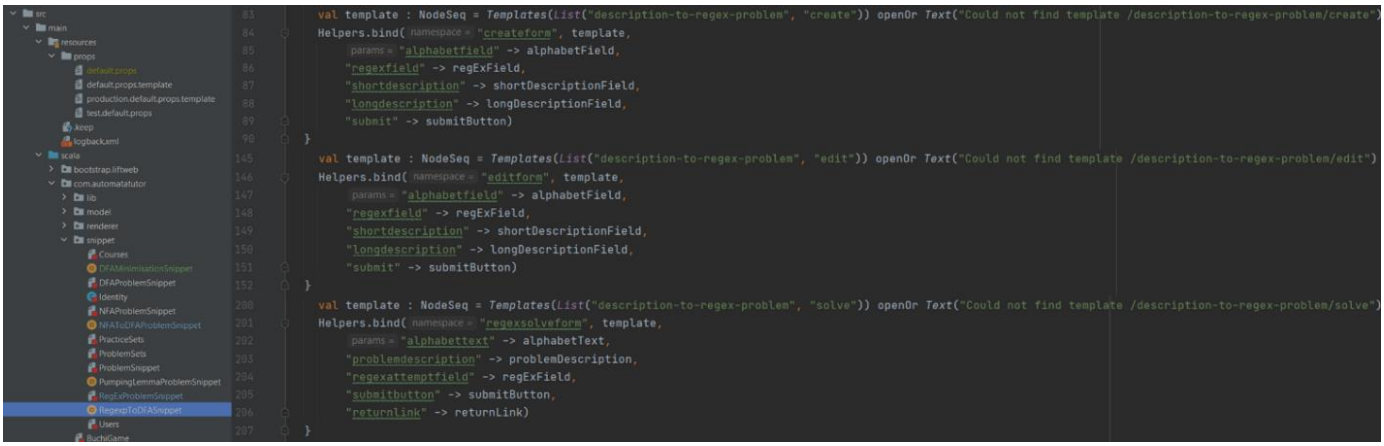


Figure 5.26: References to HTML in Regex to DFA Snippet

General Backend Implementation

The backend contains a class called “Service1” which implements a web service. This is a website that receives function calls over HTTP. “Service1” contains methods that compute feedback for given problems and their solution attempts. These methods are prefixed with “[WebMethod]” to tell the .Net web framework to make them callable over HTTP. For a new problem type, a new web method needs to be created.

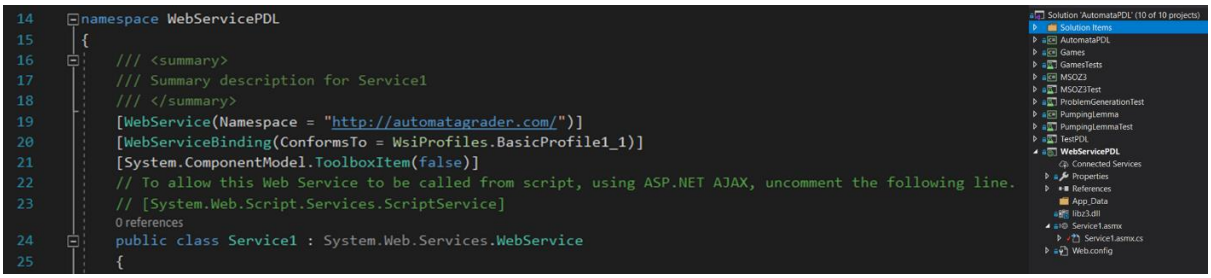


Figure 5.27: Service1

The backend makes use of a Microsoft library called Automata (Veanes, et al., 2020). This takes care of the low-level computation and includes useful methods for comparing Automata and more. This version of Automata Tutor uses an ancient version of the library. To reverse engineer it, it is necessary to search back through git history, and checkout older versions of the code that appear to have the same functions that the Automata Tutor is calling.

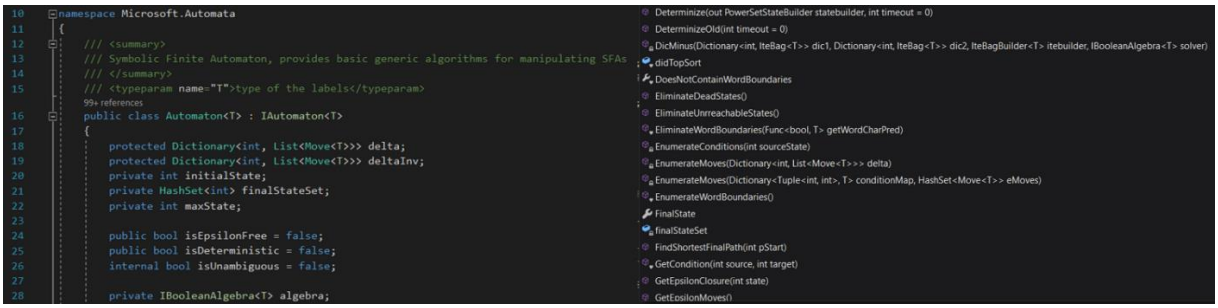


Figure 5.28: Microsoft Automata library and some of its methods

The general compute feedback pattern is to take the posed problem and the student’s attempt and convert them from xml into Automaton objects. It then solves the posed problem itself and compares it with the student attempt before generating feedback to return to the frontend. Automaton objects are part of the Microsoft library.

The backend contains utility classes which contain useful methods like “parseDFAFromXML”. There is one for DFA and NFA.

Once the Automata in question are converted into Automaton objects, the Microsoft library can be used to solve the problem with methods like “CheckDeterminism”, “Minimize”, etc.

To generate feedback there two classes: DFAGrading and NFAGrading. They take the problem solution and student attempt, and compare the two, generating a grade based on how correct the attempt is. In addition to generating a grade, these classes also generate English language feedback (hints) for the student.

DFA Minimisation Grading

The Service1 method for the NFA to DFA problem type can be duplicated and refactored for DFA Minimisation. This is because under the standard grading pattern they are theoretically similar. NFA to DFA takes two Finite Automata: the NFA in question and the student's DFA solution attempt. It then converts the NFA to a DFA and compares it with the student's solution attempt. The DFA Minimisation grading method takes a DFA instead of an NFA and minimises it before comparing it with the student attempt solution.

```
[WebMethod]
public XElement ComputeFeedbackDfaMinimisation(XElement dfaCorrectDesc, XElement dfaAttemptDesc,
XElement maxGrade)
        {
            // Turns xml into string
            #region Check if item is in cache
            StringBuilder key = new StringBuilder();
            key.Append("feedNFADFA");
            key.Append(dfaCorrectDesc.ToString());
            key.Append(dfaAttemptDesc.ToString());
            string keystr = key.ToString();

            #endregion

            // Specialised boolen algebra solver for BDDs
            CharSetSolver solver = new CharSetSolver(BitWidth.BV64);


            var dfaCorrectPair = DFAUtilities.parseDFAFromXML(dfaCorrectDesc, solver);

            dfaCorrectPair.Second.CheckDeterminism(solver);

            // Calculate correct answer from question
            var dfaCorrect =
dfaCorrectPair.Second.RemoveEpsilons(solver.MkOr).Determinize(solver).Minimize(solver);
            //(extract student's attempt from xml format)
            var dfaAttemptPair = DFAUtilities.parseDFAFromXML(dfaAttemptDesc, solver);
            dfaAttemptPair.Second.CheckDeterminism(solver);

            var level = FeedbackLevel.Hint;

            var maxG = int.Parse(maxGrade.Value);

            //Compute feedback
            var feedbackGrade = DFAGrading.GetGrade(dfaCorrect, dfaAttemptPair.Second,
dfaCorrectPair.First, solver, 1500, maxG, level, false, false, false, true);

            //Pretty print feedback
            var feedString = "<ul>";
            foreach (var feed in feedbackGrade.Second)
                feedString += string.Format("<li>{0}</li>", feed);
            feedString += "</ul>";

            var output = string.Format("<div><grade>{0}</grade><feedString>{1}</feedString></div>",
feedbackGrade.First, feedString);

            XElement outXML = XElement.Parse(output);

            return outXML;
        }
```

Figure 5.29: Compute Feedback DFA Minimisation code

"ComputeFeedbackDFAMinimisation" takes the DFA from the question and the student's attempt solution as parameters. "DFAUtilities" contains a method called "parseDFAFromXML". This takes an XML description of an Automaton and a Boolean algebra solver and returns a pair containing the alphabet and the same Automaton object from the Automata library. This object models the Automaton as a BDD.

Automaton objects initialise by calling a "Create" method. This takes start, final states and moves. It tests properties about the Automaton and sets object attributes. One of particular interest during implementation was "isDeterministic". Whilst debugging, contrary to expectation "isDeterministic" was returning false. This was an issue because both Automaton objects being worked on needed to be deterministic for subsequent operations to work. To test for determinism of an Automaton, the Create method iterates over moves and checks there are less than 2 transitions for each symbol in the alphabet. However, this was going wrong somewhere which is probably because the Automata library was still in development when this version of Automata Tutor was released. To fix the issue there was a method called "Determinize" which manually set this attribute to true, so the rest of the grading could

commence. Interestingly, the “Determinize” method was also seen in use by the creators. Perhaps this was a common fix for a known bug at the time of original development.

Automaton objects are created for the DFA question and the student’s solution attempt. To solve the question DFA, “RemoveEpsilons”, “Determinize” and “Minimize” methods are called. The student attempt is left unminimized.

To compute feedback, a “DFAGrading” method: “GetGrade” is called. This takes the correct DFA and the attempt solution and calculates how similar they are. It then generates feedback and returns it with a grade. Originally, this method minimised the correct DFA as well as the solution attempt. This was a problem because for a DFA minimisation problem, the solution attempt needs to be unminimized so it can be compared with the minimised solution. Fixing this was a challenge because modifications needed to be made to the “DFAGrading” class without affecting other parts of the system.

```
// By default student's attempt is not minimised
var dfaAttemptMin = dfaAttempt;
if (!isMinimiseProblem)
{   // if this is not grading for a minimisation problem, minimise it
       dfaAttemptMin = dfaAttemptMin.Determinize(solver).Minimize(solver);
}
//Compute minimized version of DFAs
var dfaGoalMin = dfaGoal.Determinize(solver).Minimize(solver);


#region check whether the attempt is equivalent to the solution as well as if the number of states
is the same.
// Other than minimisation problems, every dfa attempt will be minimised.
// Therefore, correct dfa attempts will always have the same number of states as the minimised dfa
goal
// i.e. Assuming the Atuomaton Library minimsation algorithm works properly,
// there will never be an instance where the dfa attempt is equivalent to the dfa goal but with a
differing number of states. (except when grading minimisation problems)
if (dfaGoalMin.IsEquivalentWith(dfaAttemptMin, solver) && dfaGoalMin.StateCount ==
dfaAttemptMin.StateCount)
{
       Console.WriteLine("Correct");
       feedList.Add(new StringFeedback(level, StringFeedbackType.Correct, al, solver));
       return new Pair<int, IEnumerable<DFAFeedback>>(maxGrade, feedList);
}
// if this is a minimisation problem and the user attempt is accepts the same language but has not
been minimised give meaninful feedback
else if (isMinimiseProblem && dfaGoalMin.IsEquivalentWith(dfaAttemptMin, solver) &&
!(dfaGoalMin.StateCount == dfaAttemptMin.StateCount))
{
       Console.WriteLine("Not Minimised");
       feedList.Add(new StringFeedback(level, StringFeedbackType.Minimisation, al, solver));
       return new Pair<int, IEnumerable<DFAFeedback>>((int)Math.Round((double)maxGrade / 4),
feedList);
}
#endregion
```

Figure 5.30: Modifications to “DFAGrading”

“GetGrade” has many parameters. It is tedious to fill every required parameter every time it gets called. Hence the developers implemented many variants of the “GetGrade” method. One main version of the method was the “real” version that took the maximum number of parameters and did all the work. Simpler versions of the method took a smaller number of input parameters and called the main version, providing sensible default values for input parameters they did not receive themselves.

To resolve the issue of unwanted automatic minimisation of student solution attempts a new parameter was added: “isMinimiseProblem”. This was a simple Boolean that allowed the automatic minimization of student answers to be turned into optional behaviour.

All the variants of the “DFAGrading” method had to be updated to handle the new parameter. By default, “isMinimiseProblem” was always set to false. This ensured that all pre-existing code continued to behave as before. With this in place, it was possible for the new “ComputeFeedbackDFAMinmisation” to call “DFAGrading” with “isMinimiseProblem” set to true. For such problems, the student’s unminimized solution attempt could now be compared with the ideal solution, to see if the student had managed to construct the minimized DFA on their own.

```
if (dfaGoalMin.IsEquivalentWith(dfaAttemptMin, solver) && dfaGoalMin.StateCount ==
dfaAttemptMin.StateCount)
{
	Console.WriteLine("Correct");
	feedList.Add(new StringFeedback(level, StringFeedbackType.Correct, al, solver));
	return new Pair<int, IEnumerable<DFAFeedback>>(maxGrade, feedList);
}
// if this is a minimisation problem and the user attempt is accepts the same language but has not
been minimised give meaninful feedback
else if (isMinimiseProblem && dfaGoalMin.IsEquivalentWith(dfaAttemptMin, solver) &&
!(dfaGoalMin.StateCount == dfaAttemptMin.StateCount))
{
	Console.WriteLine("Not Minimised");
	feedList.Add(new StringFeedback(level, StringFeedbackType.Minimisation, al, solver));
	return new Pair<int, IEnumerable<DFAFeedback>>((int)Math.Round((double)maxGrade / 4),
feedList);
}
```

Figure 5.31: Checking for Equivalence

The diagram above is the actual implementation. By default, the student attempt solution is not minimised. If the "isMinimiseProblem" is false, then it is minimised. All problem types (except DFA Minimisation) have "isMinimiseProblem" automatically set to false when calling the "GetGrade" method, so they are unaffected by my modification.

Now the system checks whether the correct DFA and the solution attempt are equivalent and if they have the same number of states. In Automata Tutor, equivalence means the two Automata accept the same language. If this condition is met, then standard feedback for a correct solution is returned. This if statement can be used for non-minimisation problems as well as minimisation problems because, other than minimisation problems, every attempt solution is minimised by default. Therefore, correct solution attempts will always have the same number of states as the minimised correct DFA. In the case of minimisation problems, if the solution attempt (which has been left unminimized) has the same number of states as the correct minimized DFA and is equivalent then this condition will be met. This all assumes the Automata library minimisation algorithm works properly. There should never be an instance where "correct" feedback is given where the attempt solution is equivalent to the correct DFA but with a differing number of states.

To generate feedback there is a "DFAFeedback" class which has several child classes representing different types of feedback e.g., "DFAEDFeedback". Normally when a student gets a question right, "StringFeedback" is used to output a message saying "Correct". "StringFeedback" generates the most basic kind of feedback. If a student gets a question wrong, then feedback is generated based on a metric of how close the student was to the correct answer. There are other types of feedback that, through a series of intelligent selections, output problem specific hints.

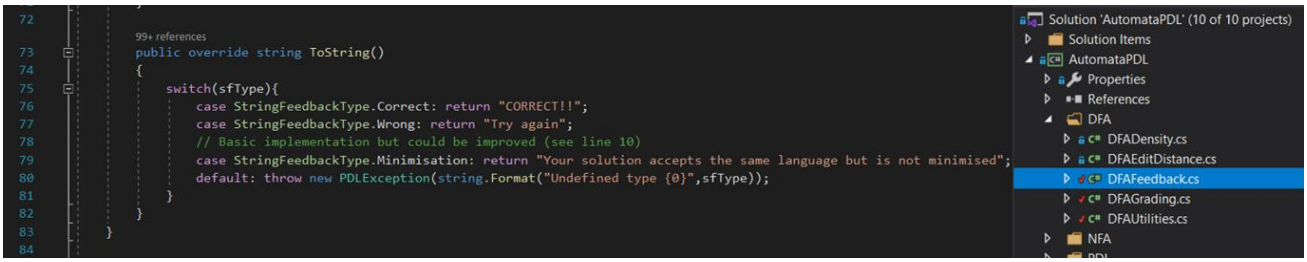


Figure 5.32: Generating minimisation feedback

In a minimisation problem, if the solution attempt and correct DFA are equivalent but the solution attempt is not fully minimised then custom feedback needs to be given. This was implemented in the form of simple "StringFeedback" that tells the user their solution accepts the same language but is not minimised.

It would have been possible to make this feedback more intelligent since it is possible to make a DFA smaller without fully minimising it. There could have been a selection process to generate feedback based on how close the solution attempt is to the minimal solution. However, understanding the system and getting DFA minimisation to work in the first place took longer than anticipated. It was not feasible to dedicate time to making this work if a second problem type was to be added before the demo.

If the correct DFA and solution attempt do not accept the same language, then the standard feedback procedure for inequivalent Automata is applied. This will inform the user of why their solution is wrong and offer hints for correcting it.

Regular Expression to DFA Grading

Regular Expression to DFA grading was more straight forward to implement because relative to other problem types in Automata Tutor it is more conventional. This meant delving into “DFAGrading” and modifying it was unnecessary.

“ComputeFeedbackRegexpToDFA” takes the question Regular Expression and DFA solution attempt as parameters and converts them to Automaton objects using the same methods described in the DFA Minimisation problem type. Note that the Automaton library can represent Regular Expressions as Automaton Objects which effectively solves the question. This was actually a factor in choosing to implement a Regular Expression to DFA problem type over the initial plan to implement probabilistic parsing, which would not have been so straight forward.

The “GetGrade” method is then called from “DFAGrading”, passing the solved DFA and the solution attempt. Feedback is returned using the standard procedure.

```
public XElement ComputeFeedbackRegexpToDFA(XElement regexCorrectDesc, XElement dfaAttemptDesc,
XElement alphabet, XElement maxGrade, XElement feedbackLevel, XElement enabledFeedbacks)
{
	#region Check if item is in cache
	StringBuilder key = new StringBuilder();
	key.Append("feed");
	key.Append(regexCorrectDesc.ToString());
	key.Append(dfaAttemptDesc.ToString());
	key.Append(feedbackLevel.ToString());
	key.Append(enabledFeedbacks.ToString());
	string keystr = key.ToString();

	var cachedValue = HttpContext.Current.Cache.Get(key.ToString());
	if (cachedValue != null)
	{
		HttpContext.Current.Cache.Remove(keystr);
		HttpContext.Current.Cache.Add(keystr, cachedValue, null,
System.Web.Caching.Cache.NoAbsoluteExpiration, TimeSpan.FromDays(30),
System.Web.Caching.CacheItemPriority.Normal, null);
		return (XElement)cachedValue;
	}
	#endregion

	CharSetSolver solver = new CharSetSolver(BitWidth.BV64);

	//Read input
	var dfaCorrectPair = DFAUtilities.parseRegexFromXML(regexCorrectDesc, alphabet, solver);
	var dfaAttemptPair = DFAUtilities.parseDFAFromXML(dfaAttemptDesc, solver);

	var level = (FeedbackLevel)Enum.Parse(typeof(FeedbackLevel), feedbackLevel.Value, true);
	var enabList = (enabledFeedbacks.Value).Split(',').ToList<String>();
	//bool dfaedit = enabList.Contains("dfaedit"), moseledit = enabList.Contains("moseledit"),
density = enabList.Contains("density");
	bool dfaedit = true, moseledit = true, density = true;

	var maxG = int.Parse(maxGrade.Value);

	//Compute feedback
	var feedbackGrade = DFAGrading.GetGrade(dfaCorrectPair.Second, dfaAttemptPair.Second,
dfaCorrectPair.First, solver, 1500, maxG, level, dfaedit, density, moseledit, false); //***

	//Pretty print feedback
	var feedString = "<ul>";
	foreach (var feed in feedbackGrade.Second)
	{
		feedString += string.Format("<li>{0}</li>", feed);
		break;
	}
	feedString += "</ul>";

	//var output =
string.Format("<result><grade>{0}</grade><feedString>{1}</feedString></result>",
feedbackGrade.First, feedString);
	var outXML = new XElement("result",
			new XElement("grade", feedbackGrade.First),
			new XElement("feedString", XElement.Parse(feedString)));
	//XElement outXML = XElement.Parse(output);
	//Add this element to chace and return it
	HttpContext.Current.Cache.Add(key.ToString(), outXML, null,
System.Web.Caching.Cache.NoAbsoluteExpiration, TimeSpan.FromDays(30),
System.Web.Caching.CacheItemPriority.Normal, null);

	return outXML;
}
```

Figure 5.33: Compute Feedback Regular Expression to DFA code

Frontend and Backend Communication

Finally, the frontend needs to communicate with the backend. A crucial middle tier class is “SOAPConnection”. This is called by Snippets to make requests to the grading engine when a user submits a solution attempt.

New problem types need to have their own methods in this class. It needs to call a backend method in service1.asmx and pass all the necessary parameters. This will always include the question and the student’s attempted solution. At this stage, the new problem type should already have a service1.asmx method to call.

For the DFA Minimisation problem type there is a method in “SOAPConnection” called “getDFAMinimisationFeedback”. This calls the “ComputeFeedbackDFAMinimisation” from method in the grading engine.

Figure 5.34: “SOAPConnection” for DFA Minimisation

Similarly, “getRegexToDFAFeedback” calls “ComputeFeedbackRegexpToDFA” in the backend.

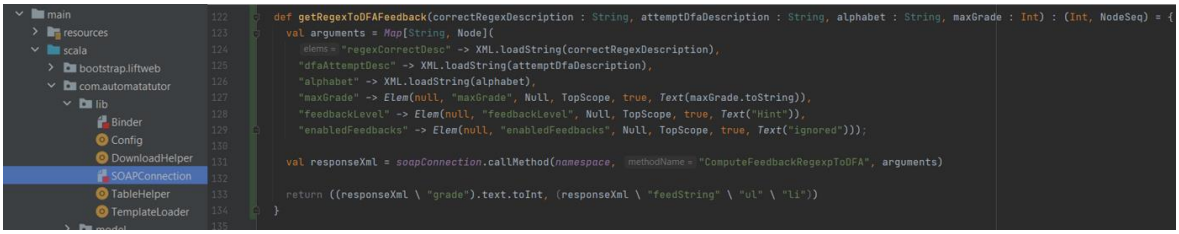


Figure 5.35: “SOAPConnection” for Regular Expression to DFA

Problems with Regular Expression to DFA

Implementation of this problem type seemed more straight forward because of the experience gained by successfully implementing DFA minimisation. There was less uncertainty and more confidence. However, although most aspects of the implementation are complete, the solve functionality in the Snippet does not work properly. There are also underlying issues. Since this problem type was theoretically similar with English to Regular Expression and English to DFA, code was used from both. This was to save time so there would be two working exercise types for the demo. The implementation is rushed and there seems to be problems when “RegExpToDFASnippet” attempts to communicate with the grading engine. The issue is most likely with the problem model. To fix it, careful evaluation of how code has been reused from other parts of the system and whether they are compatible would need to be undertaken. If there was more time implementation could be done from scratch.

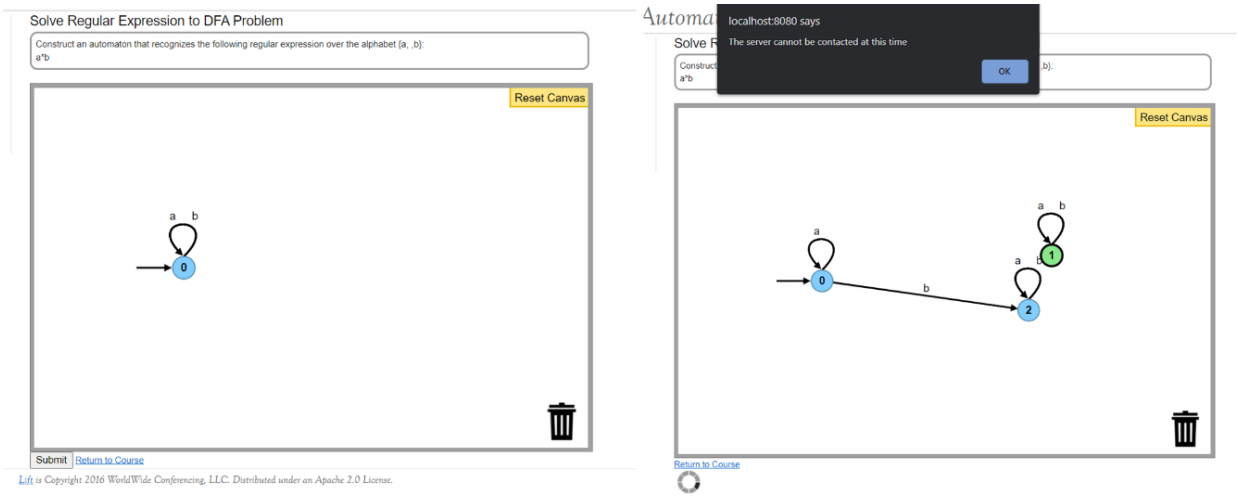


Figure 5.36: Demonstration of solving Regular Expression to DFA Problem

A positive to take from this is that this is an example of why copy/paste coding is generally frowned upon as well as the difficulties of Software Engineering under time constraints.

Testing

This project was about extending an existing application which had already been tried and tested. Regression testing was sufficient in most places. This is where functional code is re-run after making additions to ensure the additions have not broken anything.

Regression Testing

Frontend

The frontend had very few pre-existing automated regression tests. It seems the main method for testing was to view changes when running the webapp. More regression tests could have been added but the webapp seemed to work as expected and given the time constraints it seemed like a non-essential task.

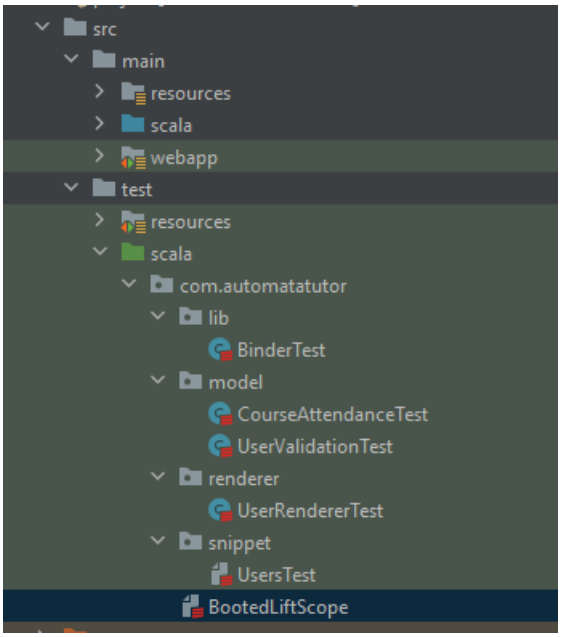


Figure 6.0: Frontend testing directory

Backend

The backend included many regression tests which were used to ensure changes still worked as expected. In fact, 11 of the tests made calls to the “GetGrade” method which had an additional parameter because of the DFA Minimisation problem type implementation. These calls were updated to include the new parameter. Running the tests produced good results as seen in the figure below.

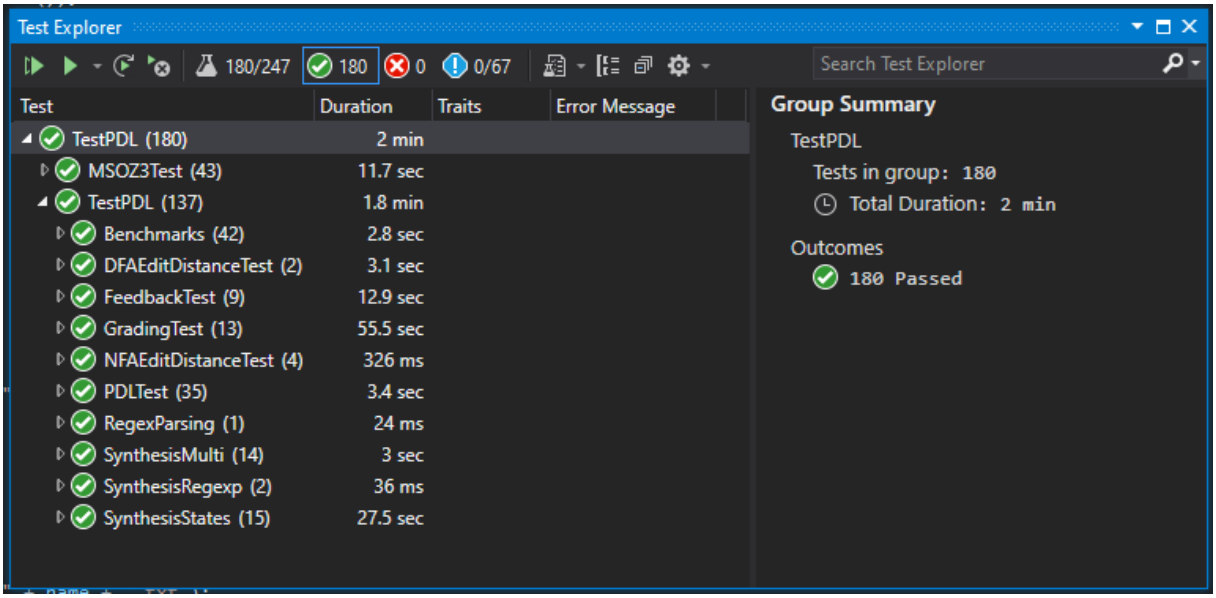


Figure 6.1: Backend regression tests

Verifying Database Persistence DFA Minimisation

It was also important to test the new problem types worked. This was done by verifying database persistence when creating, editing, or solving new problem types. In addition, testing the grading engine worked properly by assessing the feedback on a multitude of questions and solution attempts as well as the quality of the feedback returned.

Create & Edit

This tests whether a DFA Minimisation problem appears in the database when creating it in the webapp. It also tests whether editing the problem updates in the database. Automata are stored as xml. As seen in the screenshot below, everything works as expected.

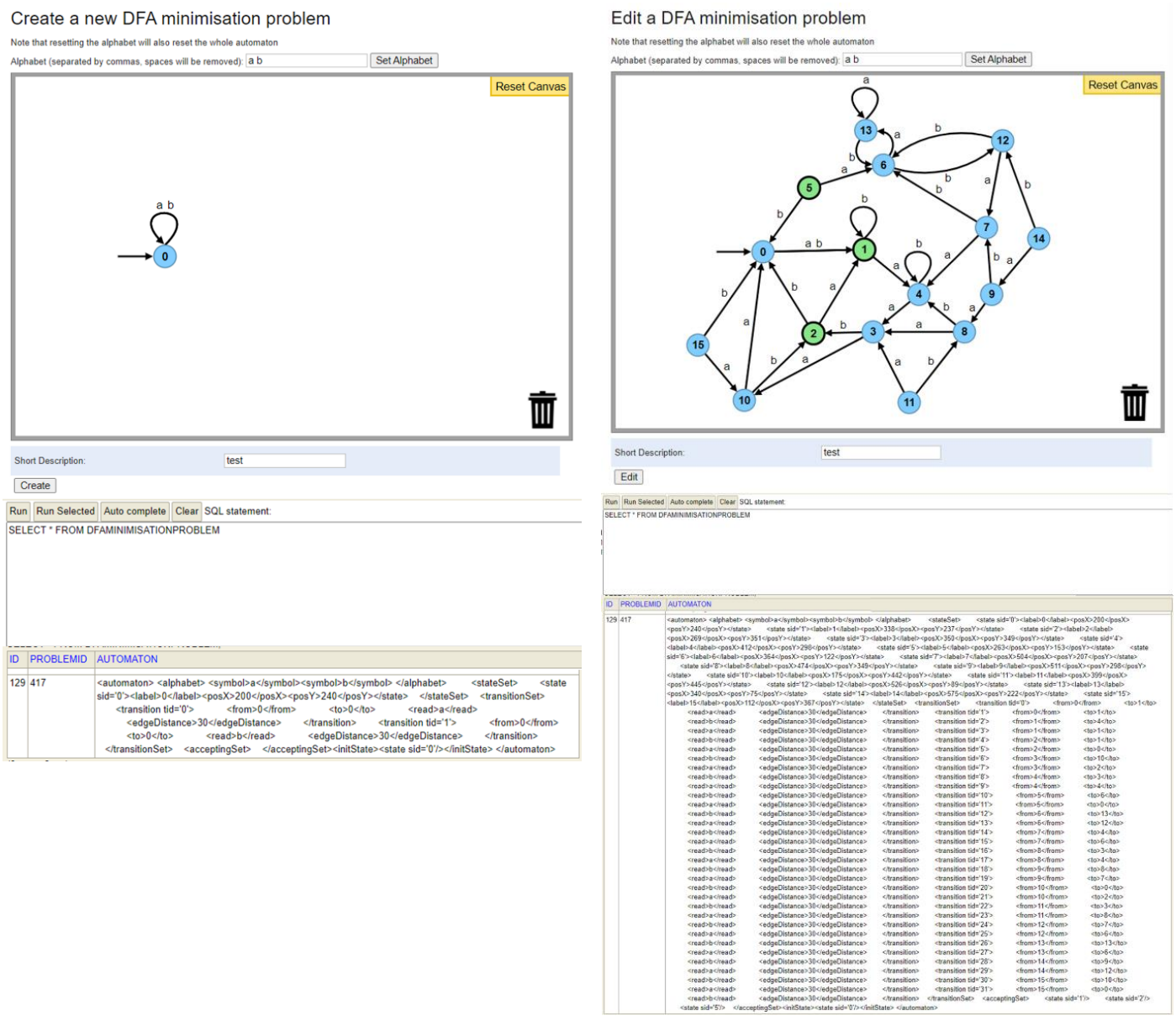


Figure 6.2: create and edit tests

Solve

This tests whether a student's solution attempt is stored in the database. This data is spread over two tables. One stores the grade and time whilst the other stores the actual DFA the user submitted. They are linked by a common ID. As seen in the screenshot below, everything works fine.

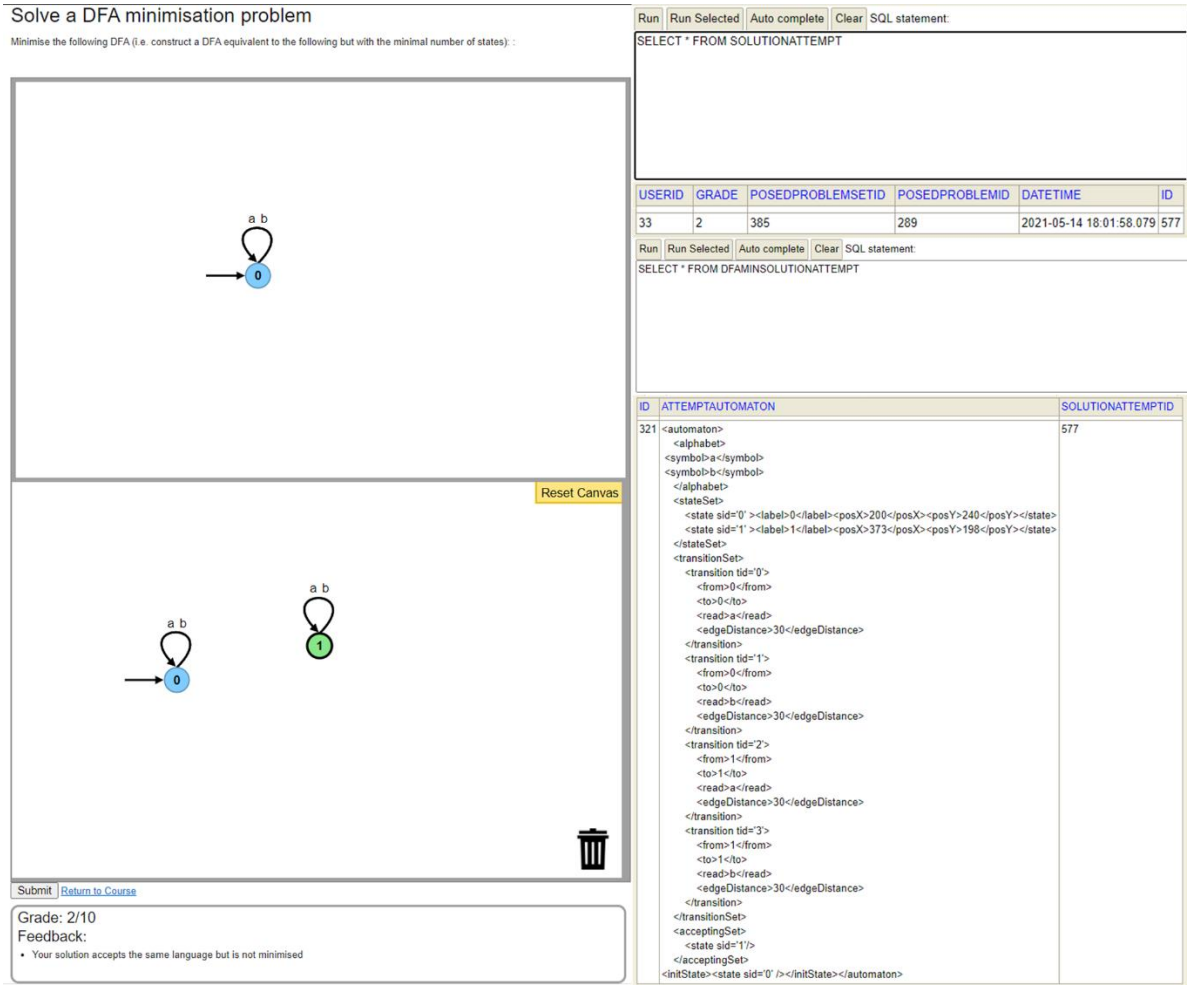


Figure 6.3: solve test

Delete

This tests whether deleting a problem updates the database appropriately. When this problem was created there were 9 rows in the database table. As seen in the screenshot below this has now been reduced to 8 when an admin deleted the problem.

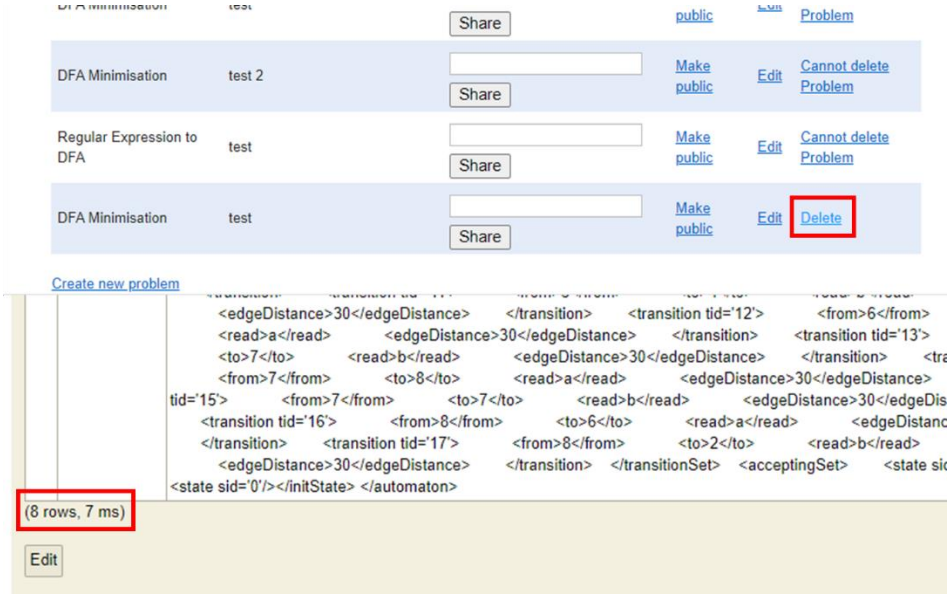


Figure 6.4: on delete test

Verifying Database Persistence Regular Expression to DFA

Create

This tests whether a Regular Expression to DFA problem appears in the database when creating it in the webapp. As seen in the screenshot below, everything works as expected.

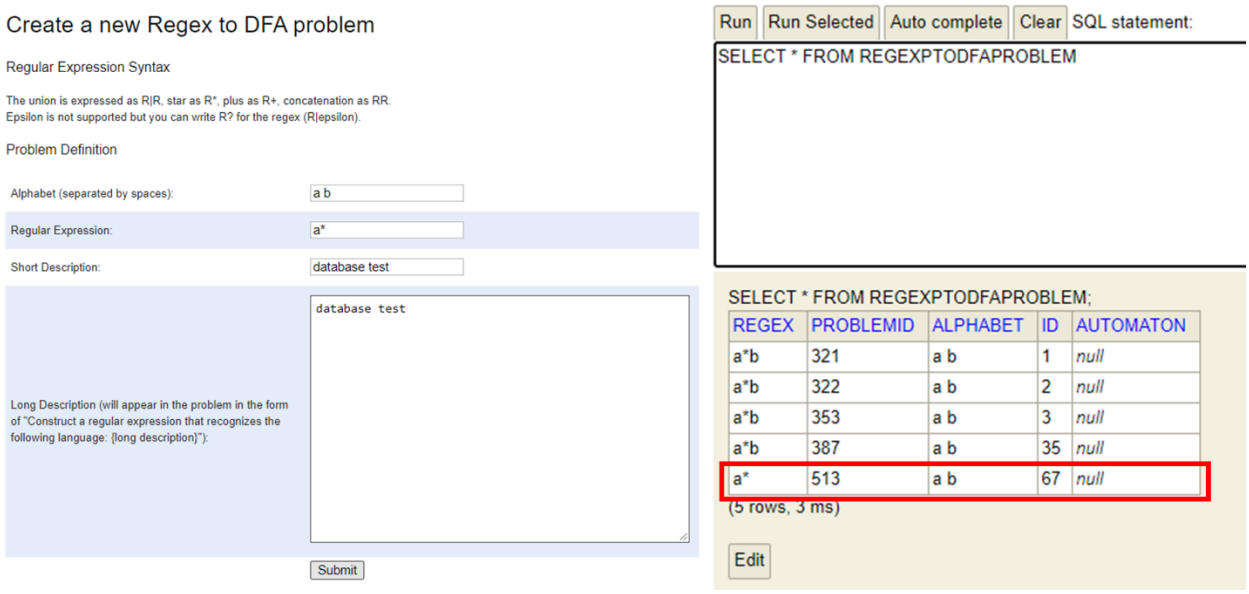


Figure 6.5: create test

Edit

This tests if the database successfully updates when an admin edits a problem. As seen in the screenshot below, everything works as expected.

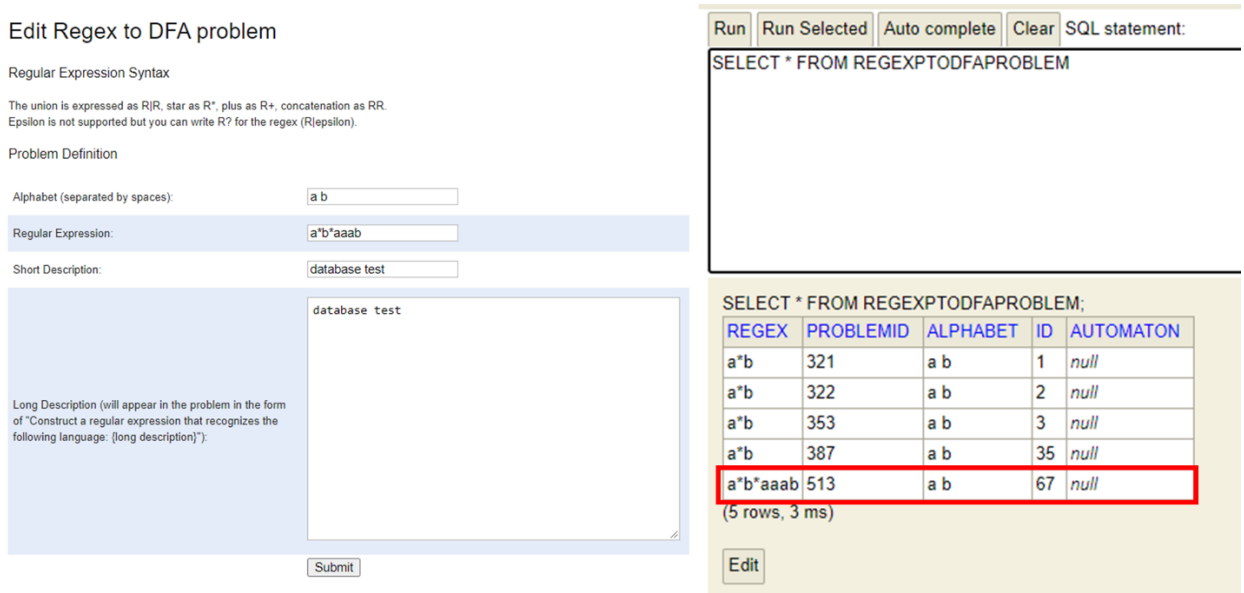


Figure 6.6: edit test

Solve

This tests if a student's solution attempt is stored in the database. Unfortunately, as seen in the screenshot below, this does not work because the solve functionality has issues with its implementation. Since a grade and feedback is never returned the submission cannot be completed so nothing will be stored in the database. Hence why the Regular Expression to DFA Solution Attempt table is empty.

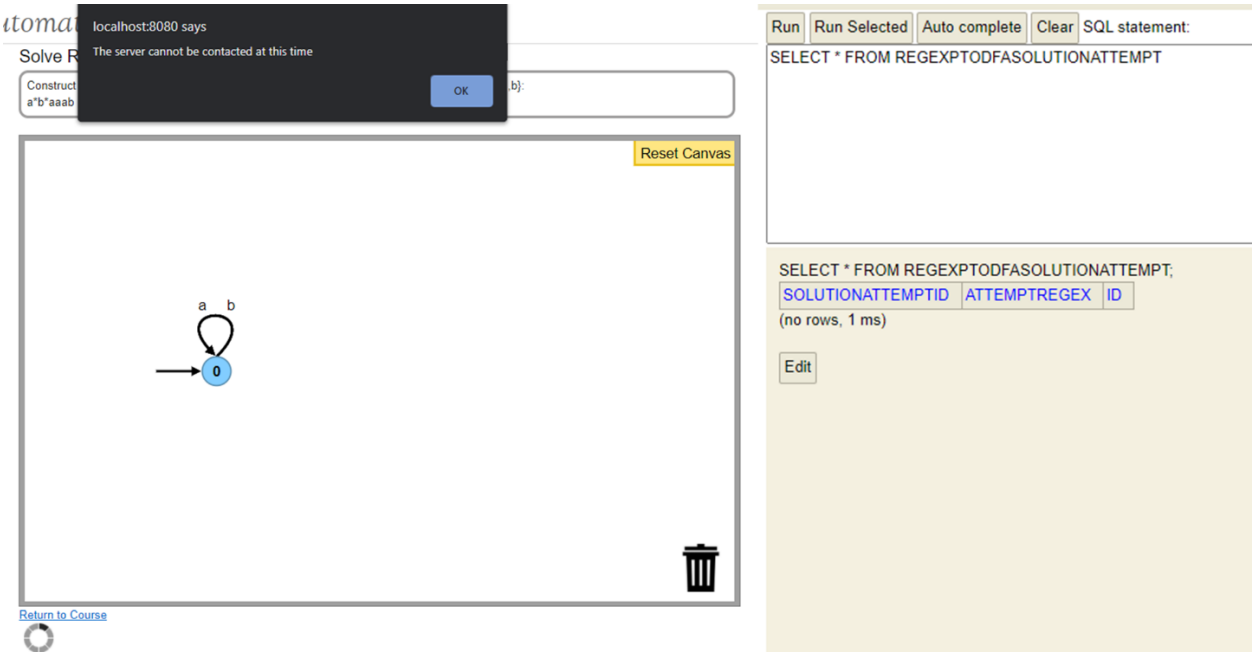


Figure 6.7: solve test

Delete

Similarly, deleting a Regular Expression to DFA problem does not update the database indicating that something is wrong with the delete functionality too. As seen in the screenshot below, the row in the database table representing the problem added did not disappear as expected.

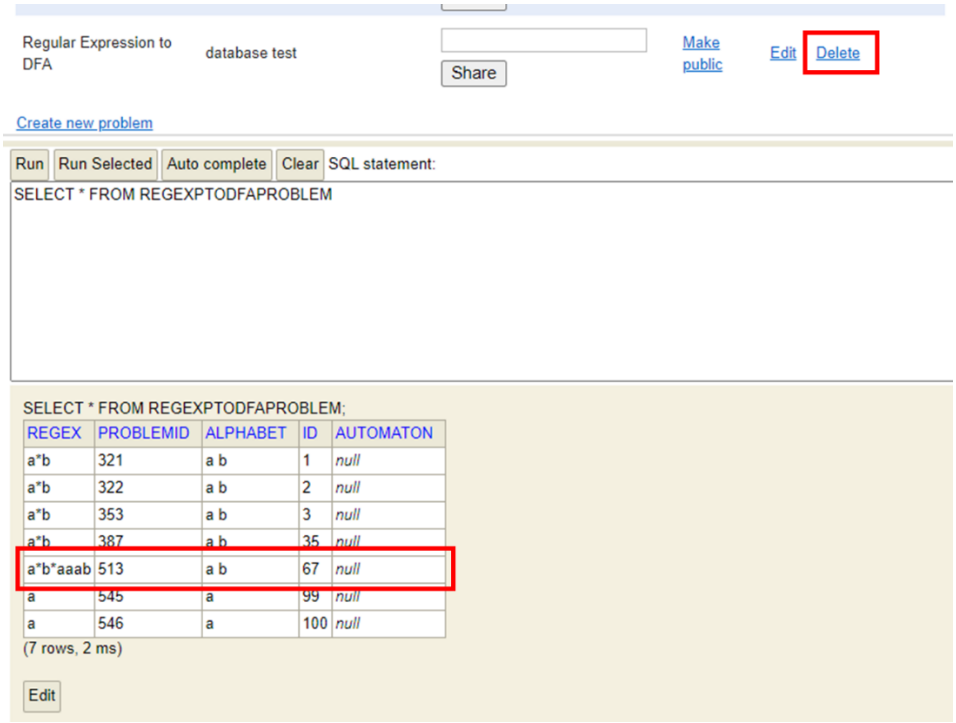
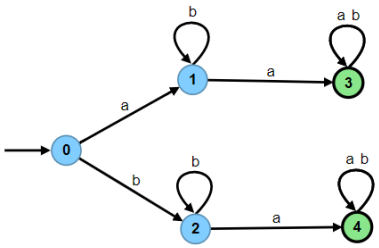
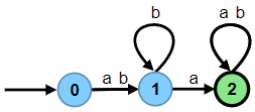
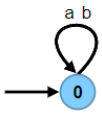


Figure 6.8: on delete test

Grades and Feedback DFA Minimisation

Test 1 – Bad Solution Attempt

This tests feedback when a student gets the question wrong. The students Automaton is not even close to the correct answer. Automata Tutor grades the work 0/10 and gives an appropriate hint as to why the student's solution is wrong.

Posed Problem	Correct Answer	Solution Attempt
		

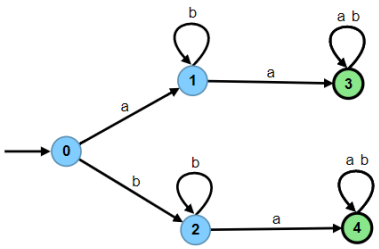
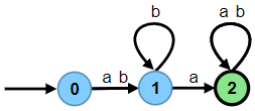
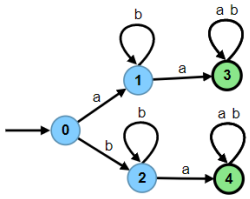
Grade: 0/10
Feedback:

- Your solution is not correct on this set of strings:

{ *s* | *ba* appears in *s* at least *once* }

Test 2 – Same DFA as Question

This tests feedback when a student inputs the same minimizable DFA given in the question. Automata Tutor awards a harsh grade since despite accepting the language, the student has not done any minimisation.

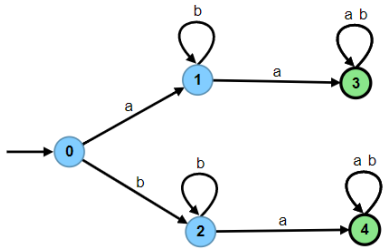
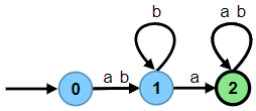
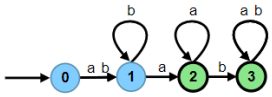
Posed Problem	Correct Answer	Solution Attempt
		

Grade: 2/10
Feedback:

- Your solution accepts the same language but is not minimised

Test 3 – Not Fully Minimised

This tests feedback when a student does minimise the DFA given in the question but does not provide the most minimized solution. The feedback is the same as not minimising the DFA at all. This is because the implementation of feedback, simply tests whether the minimal solution and student attempt accept the same language and have the same number of states. Whereas other problem types generate a metric of how similar the solution attempt and correct solution are and generate feedback based on this.

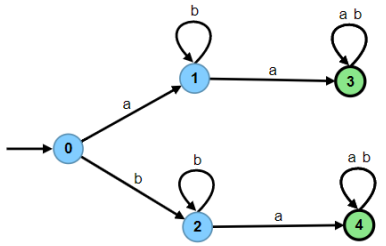
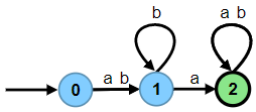
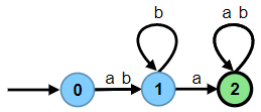
Posed Problem	Correct Answer	Solution Attempt
		

Grade: 2/10
Feedback:

- Your solution accepts the same language but is not minimised

Test 4 – Correct Solution Attempt

This tests feedback when a student gets the question right. Automata Tutor awards full marks and tells the student their solution is “CORRECT!!”.

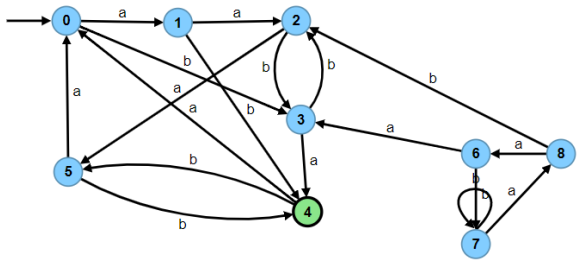
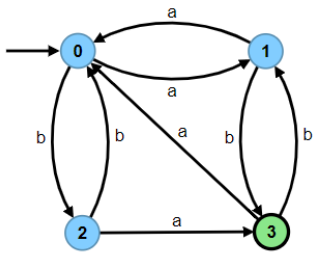
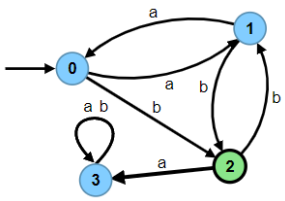
Posed Problem	Correct Answer	Solution Attempt
		

Grade: 10/10
Feedback:

- CORRECT!!

Test 5 – Complicated Automaton Bad Solution Attempt

The Automaton posed in this problem tests whether Automata Tutor can handle complicated Automata. This test reveals what happens if the student gets the question wrong. Automata Tutor returns the expected feedback.

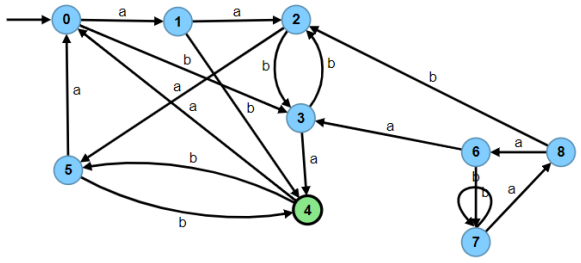
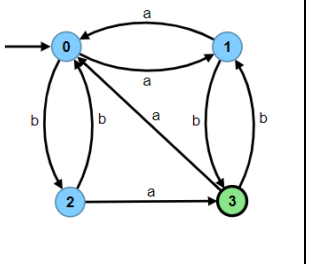
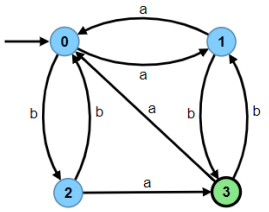
Posed Problem	Correct Answer	Solution Attempt
		

Grade: 0/10
Feedback:

- Your solution does not accept the string '*ba*' while the correct solution does.

Test 6 – Highlighting a Known Bug

This tests feedback when a student gets the answer correct for a complicated Automaton question. Automata Tutor returned unexpected feedback. Despite being the correct answer, Automata Tutor grades this harshly and claims the solution accepts the same language but is not minimised.

Posed Problem	Correct Answer	Solution Attempt
		

Grade: 2/10
Feedback:

- Your solution accepts the same language but is not minimised

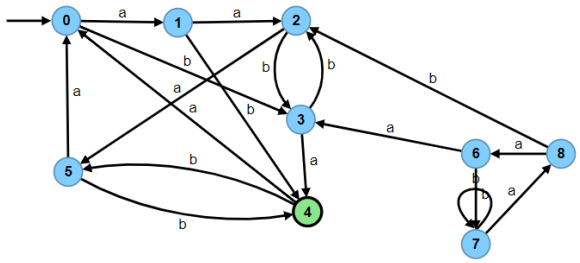
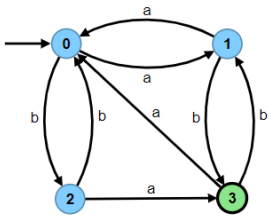
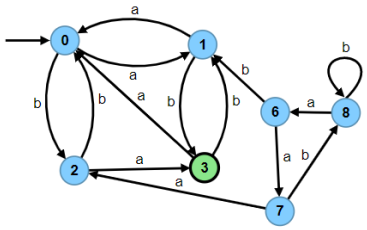
This occurs because when the Microsoft Automata library minimises Automata, it fails to recognise and remove redundant states. In the posed problem, states 6, 7 and 8 are unreachable so they can be completely removed. Since the grading engine leaves these states in its minimal solution, and the solution attempt needs the same number of states to be classed as correct, the actual correct solution is given the wrong feedback. i.e., it has less states than Automata Tutor's minimal solution.

A quick fix could have been to accept solutions that have less than or equal the number of states of the grading engine calculated minimised solution. However, this creates another problem: leaving redundant states in your answer is given a grade and feedback for a correct answer.

Another way to fix this issue would have been to update Automata Tutor to use the latest version of the Automata library which most likely has this problem resolved. However, this probably would have involved adjusting many parts of the grading engine to make it compatible which seemed like an infeasible task given the time limits. It also would have been pointless because the latest version of Automata Tutor already has these adjustments and works with a later version of the library. Alternatively, the version of the Automata library used by this version of Automata Tutor could have been updated, but this also would have been complicated.

Test 7 – Minimising but Leaving Redundant States

As seen below, putting the redundant states back into the correct solution produces the feedback expected (as discussed in Test 6).

Posed Problem	Correct Answer	Solution Attempt
		

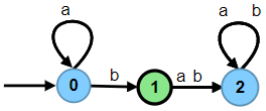
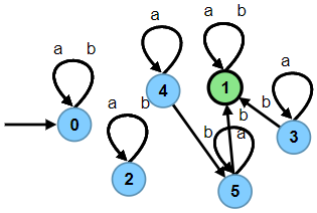
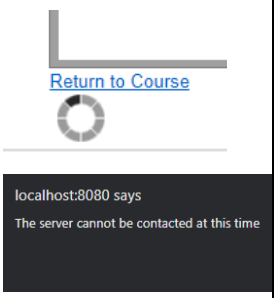
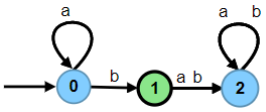
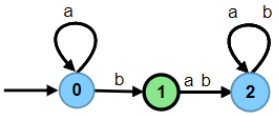
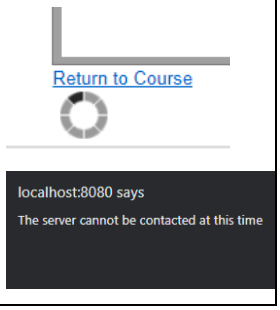
Grade: 10/10

Feedback:

- CORRECT!!

Grades and Feedback Regular Expression to DFA

Unfortunately, as previously discussed, solving a Regular Expression to DFA problem does not currently work as illustrated below.

Posed Problem	Correct Answer	Solution Attempt	Grader Response
“Construct an automaton that recognizes the following regular expression over the alphabet {a, b}: a*b”			
“Construct an automaton that recognizes the following regular expression over the alphabet {a, b}: a*b”			

Evaluation

The aim to add DFA Minimisation was successfully fulfilled. The process of adding it was effective in providing experience of Software Engineering, various technologies, and conventions, and representing Automata on modern computers. So, the aims related to learning and experience were also fulfilled.

Regular Expression to DFA was mostly implemented but left unfinished. This provided first-hand experience of time constraints and their impact on the quality of software. Although the aims related to implementing this second problem type were unfulfilled, this provided further experience of Software Engineering.

Most of the objectives were steps towards implementing the new problem types so objectives 1-7 and objective 10 were fulfilled. However, objectives 8 and 9 were only partially completed.

Conclusion

The initial expectation for this project was that it would be about writing algorithms to perform operations on Automata like minimisation. As it turned out, this work was already done by the creators. The project was really about figuring out how Automata Tutor worked and then Software Engineering.

Once the implementation was understood, the project was like plumbing. It was necessary to follow the existing patterns to make new additions, and then to make sure everything was wired up correctly.

It has been extremely satisfying to see the finished working result of DFA Minimisation. A lot has been learned about Software Engineering through the process of researching and implementing. Although, Regular Expression to DFA is unfinished this has at least yielded an experience of why Software Engineering is such a large and widely studied subject. Also, why it is common for companies and organisations in the Computer Science world to fail in delivering successful software projects (The Standish Group, 1994). There are so many conventions and approaches and architectures and ways of solving problems that creating something functional and of a high quality can often go beyond expectation and time constraints. “There is no silver bullet” (Frederick P. Brooks, 1987).

Finally, Automata Theory is an extremely interesting subject. It has been a pleasure to work with it with the added satisfaction that the new problem types may contribute to teaching it. It has also been great to research it and learn about the Microsoft Automata library.

BCS Project Criteria & Self-Reflection

“An ability to apply practical and analytical skills gained during the degree programme.”

Before undertaking this project, I had completed two modules in Software Engineering which taught me many concepts and ideas. I had a conceptual understanding of multi-tiered architecture and MVC and the sorts of technologies used in the real world but not a practical experience. This project allowed me to apply the concepts I had learned since the magnitude and breadth of the software made it a necessity. It would have been impossible to understand the system without being able to spot coding patterns or where abstractions or dependencies had been made and why. Having studied Software Engineering beforehand was also important for analysing the system and identifying which parts needed to be worked on to realise my aims as well as ensuring a high-quality implementation. This project also confirmed many of the struggles of Software Engineering discussed in the modules I completed.

Prior knowledge of Automata Theory was also useful for understanding the problem types I implemented. It was also interesting to learn about how Automata can be represented on modern computers as BDDs and why this was necessary.

General knowledge of coding, common practices, and Object-Oriented programming from various modules throughout my degree programme also proved particularly useful for the implementation of the backend. This is because making the grading engine work for DFA Minimisation was not straight forward. There were many obstacles which had solutions that were not immediately obvious.

“Innovation and/or creativity.”

Since the documentation for Automata Tutor was relatively light, I endeavoured to add to it with my own diagram of the system architecture and provide somewhat of a step-by-step strategy for adding a new problem type with more detail than the GitHub Read-Me. This can be seen in the design and implementation sections.

Creating a new problem type that was not there before involved a lot of creativity. There are so many different technologies involved in this project many of which I am not proficient in. I had to think creatively to get around this. I had to study the code to see which parts I could replicate and spot patterns and conventions to ensure I was sticking to them. I also had to spot where I lacked knowledge. It was sometimes necessary to learn how to implement bits where code could not be reused.

I also had to overcome the issue of unwanted automatic minimisation in a way that did not affect other parts of the system. For this I had to determine the best approach.

“Synthesis of information, ideas and practices to provide a quality solution together with an evaluation of that solution.”

There was a diverse array of technologies and components to this project. I had to learn everything to a standard enabling me to make an implementation of a new problem type. See Technologies, System Architecture Design, Implementation and Testing sections.

“That your project meets a real need in a wider context.”

See motivation section.

Hopefully, this document can also help developers in the future (as the Automata Tutor documentation is light) by presenting my experience and insights.

“An ability to self-manage a significant piece of work.”

Figure was my original project schedule. It has been interesting to see how initial conceptions of the important aspects and time scales changed as the project progressed. Scala was important, and I have certainly learned some basics, but realistically it did not take as long as depicted below to know enough to implement a new problem type.

A large portion of time was spent reverse engineering the system to gain an understanding of it. This was quite a gruelling process because I did not necessarily expect it and it often felt like no progress was being made which is demoralising. I had to remind myself that research was a worthy time investment even if nothing was being added to the system. If I could not implement anything within the time constraints, then at least I could contribute to extending the documentation by explaining my research and working understanding. In the end, implementation was relatively straightforward which was testament to all the things I had learned. The research paid off!

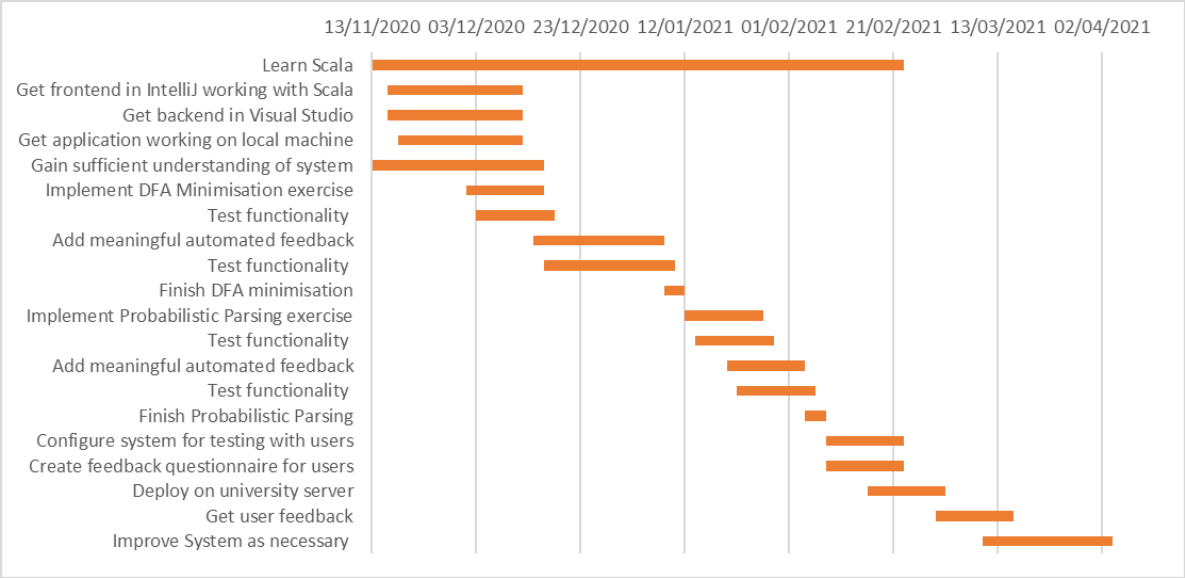


Figure 7.0: Project schedule from Detailed Proposal

“Critical self-evaluation of the process.”

When I first started, the code was extremely daunting. I underestimated how much effort goes into just understanding a system of this size. This made progress seem painfully slow to begin with because of the amount of unexpected time spent researching and studying the code. There was no straight forward path to success as I had naively expected.

Typical struggles during my project included:

- Something not working as expected the first time and spending many demoralising hours working out why. Frustratingly this was normally with things that were supposed to be menial e.g., IDE settings, installing the right version of Java for Scala etc.
- Discovering new issues and realising I had more to learn before I could begin working on the thing I set out to do.
- Investing time to learn something to later discover it was unnecessary.

This has taught me that Software Engineering can be a laborious process which requires focus and determination.

If I had to do the project again, I would advise myself not to be put off by the scale of Automata Tutor's implementation. To expect obstacles and fruitless time investments because this is part of the process of finding the right solution. Even if time is wasted learning about something that turned out to be irrelevant, at least it was still productive. Software Engineering is not easy, so it is okay to struggle.

Ultimately, I ran out of time. This is the reason my implementation of Regular Expression to DFA is rushed and has flaws. This is a situation that could have been avoided if I had more experience working on a project this size because the code would have seemed less daunting, and I would be more confident with my work. If I had the experience I have now, I probably would have more ideas about how to get started too.

If I had more time, I would have worked on the following:

- Finishing Regular Expression to DFA implementation
- Making DFA Minimisation feedback more sophisticated
- Creating more regression tests for the frontend

Overall, I am happy with what I have achieved because I feel that this project has given me some real practical, hands on, valuable experience. In addition to this I had the opportunity to extend my knowledge of Automata Theory which has been fantastic. I also feel more confident with my understanding of Software Engineering compared to when I started because of the insight I have gained with this project.

Although I have successfully implemented an exercise and almost added another, there is always more to understand about the system. Anyone using my documented experience as a guide in the future, should take into consideration my limited experience when I started. It is possible that in some aspects of the project I have undetected misconceptions because my understanding of the system was still able to produce a working result. The most reliable source are the creators of Automata Tutor.

If you are undertaking this project and attempting to add a new problem type and you are starting with a limited understanding of the system as I did, then my general advice would be to us the GitHub read me as a starting point for research. Then to identify and conform with the existing patterns and work from existing code wherever possible.

References

Carroll, P., 2015. *Relationship between C#, .NET, ASP.NET etc.* [Online]
Available at: https://softwareengineering.stackexchange.com/questions/44810/relationship-between-c-net-asp-asp-net-etc#:~:
[Accessed 14 May 2021].

Code Academy, 2021. *MVC: Model, View, Controller.* [Online]
Available at: https://www.codecademy.com/articles/mvc
[Accessed 14 May 2021].

Cox, L. K., 2020. *Web Design 101: How HTML, CSS, and JavaScript Work.* [Online]
Available at: https://blog.hubspot.com/marketing/web-design-html-css-javascript
[Accessed 14 May 2021].

Curie, D. H., Jaison, J., Yadav, J. & Fiona, J. R., 2019. Analysis of Web Frameworks. *Journal of Physics: Conference Series,* Volume 1362.

D'Antoni, L. et al., 2015. How can Automatic Feedback Help Students Construct Automata?. *ACM Transactions on Computer-Human Interaction,* Volume 22.

D'Antoni, L., Weaver, M., Weinert, A. & Alur, R., 2015. Automata Tutor and What we Learned from Building an Online Teaching Tool. *The Education Column,* Volume 117.

Frederick P. Brooks, J., 1987. No Silver Bullet: Essence and Accidents of.

freeCodeCamp, 2020. *How and why to use Functional Programming in modern JavaScript.* [Online]
Available at: https://www.freecodecamp.org/news/how-and-why-to-use-functional-programming-in-modern-javascript-fda2df86ad1b/#:~:text=Advantages%20Of%20Functional%20Programming,improve%20maintainability%20of%20the%20code.
[Accessed 12 May 2021].

Friedl, J. E. F., 1997. *Mastering Regular Expressions.* 3rd ed. s.l.:O'REILLY.

GitHub, 2021. *GitHub.* [Online]
Available at: https://github.com/
[Accessed 12 May 2021].

Google, 2021. *Chrome Developers.* [Online]
Available at: https://developer.chrome.com/docs/devtools/overview/
[Accessed 13 May 2021].

h2, 2021. *H2 Database Features.* [Online]
Available at: http://www.h2database.com/html/features.html#embedded_databases
[Accessed 13 May 2021].

Hopcroft, J. E., Motwani, R. & Ullman, J. D., 1979. Finite Automata with Epsilon-Transitions. In: *Automata Theory, Languages, and Computation.* 3rd ed. s.l.:Addison-Wesley, p. 72.

Hopcroft, J. E., Motwani, R. & Ullman, J. D., 1979. *Introduction to Automata Theory, Languages, and Computation.* 3rd ed. s.l.:Addison-Wesley.

Kandhway, H. & Theraja, P., 2019. *Xebia Engineering Blog: A general introduction to build tools.* [Online]
Available at: https://medium.com/xebia-engineering/a-general-introduction-to-build-tools-9070a47ed405
[Accessed 12 May 2021].

Lift Framework, 2021. *Lift Framework Embed.* [Online]
Available at: https://www.liftweb.net/api/32/api/net/liftweb/builtin/snippet/Embed$.html
[Accessed 16 May 2021].

Long, S. C., 2008. *git.* [Online]
Available at: https://git-scm.com/
[Accessed 12 May 2021].

Martin Odersky, 2021. *Scala: Getting Started.* [Online]
Available at: https://docs.scala-lang.org/getting-started/index.html
[Accessed 14 May 2021].

Mueller, T., 2006. *H2 Database Engine.* [Online]
Available at: http://www.h2database.com/html/main.html
[Accessed 14 May 2021].

Patterson, D., 2013. *Why are English Majors Studying Computer Science?.* [Online]
Available at: https://blogs.berkeley.edu/2013/11/26/why-are-english-majors-studying-computer-science/
[Accessed 16 May 2021].

Pollak, D., 2007. *Lift.* [Online]
Available at: https://www.liftweb.net/
[Accessed 9 May 2021].

Pollak, D., 2011. *Simply Lift.* [Online]
Available at: https://simply.liftweb.net/index-3.4.html#toc-Section-3.4
[Accessed April 2021].

Pollak, D., 2011. *Simply Lift.* [Online]
Available at: https://simply.liftweb.net/index.html
[Accessed 9 May 2021].

Sisper, M., 1997. *Introduction to the Theory of Computation.* International 3rd Edition ed. s.l.:PWS Publishing.

Srivathsan, B., 2015. *Introdcution to BDDs.* [Online]
Available at: https://youtu.be/sg7Ae9crtTU
[Accessed March 2021].

Srivathsan, B., 2015. *Ordered BDDs.* [Online]
Available at: https://youtu.be/QiHhSAMur1U
[Accessed March 2021].

Srivathsan, B., 2015. *Representing Transition Systems as OBDDs.* [Online]
Available at: https://youtu.be/l6XjAWRQkcw
[Accessed March 2021].

Stack Overflow, 2011. *How do I debug Lift applications in Eclipse?.* [Online]
Available at: https://stackoverflow.com/questions/8621542/how-do-i-debug-lift-applications-in-eclipse
[Accessed 2021].

The Standish Group, 1994. *The CHAOS Report,* s.l.: s.n.

Veanes, M. & Bjørner, N., 2012. Symbolic Automata: The Toolkit. *Flanagan C., König B. (eds) Tools and Algorithms for the Construction and Analysis of Systems,* Volume 7214.

Veanes, M., D'Antoni, L. & Saarikivi, O., 2020. *AutomataDotNet/Automata.* [Online]
Available at: https://github.com/AutomataDotNet/Automata
[Accessed 13 January 2021].

Voth, G. R., Kindel, C. & Fujioka, J., 1998. *Distributed Application Development for Three-Tier Architectures: Microsoft on Windows DNA,* s.l.: Microsoft Corporation.

W3 schools, 2021. *HTML Canvas Graphics.* [Online]
Available at: https://www.w3schools.com/html/html5_canvas.asp
[Accessed 14 May 2021].

Wagner, B., 2021. *A tour of the C# Language.* [Online]
Available at: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/
[Accessed 14 May 2021].

Weinert, A. & D'Antoni, L., 2019. *Automata Tutor Frontend README.* s.l.:s.n.

Weinert, A. & D'Antoni, L., 2021. *Automata Tutor.* [Online]
Available at: https://automata-tutor.model.in.tum.de/
[Accessed May 2021].

Wikipedia, 2021. *Functional Programming.* [Online]
Available at: https://en.wikipedia.org/wiki/Functional_programming
[Accessed 12 May 2021].

Wikipedia, 2021. *Multitier architecture.* [Online]
Available at: https://en.wikipedia.org/wiki/Multitier_architecture
[Accessed 14 May 2021].

Wikipedia, 2021. *Web framework.* [Online]
Available at: https://en.wikipedia.org/wiki/Web_framework
[Accessed 12 May 2021].

Wojtczak, D., 2019. *Decision, Computation and Language: DFAs, NFAs, εNFAs, and Regular Expressions: General Method.* Liverpool: University of Liverpool.