Automated Exercises
Final Report

CS 4624

4/26/19

Virginia Tech, Blacksburg
Mostafa Mohammed

Clayton Cunningham
Jacob Mokuvos
Mingchi Li
Shuhao Zhou
Shengwei Liu

Table of Contents

Table of Figures

Summary

The goal of this project is to create an automated assessment exercise framework which will allow instructors to build a number of different exercises for Formal Languages course by uploading JFLAP file to OpenDSA textbook. The project will impact both instructors and students. Instructors will use it to build different exercises to students without considering the time and effort to grade these exercises manually. Students will use these exercises to practice more on different topics as available. The final product will eventually have to complete generating exercises, auto-grade exercises and store students answers and grades in OpenDSA database.

To complete the project, we will need to utilize some basic web design language, such as HTML, JavaScript, We had complete being able to generate, complete, and grade the exercises of all the topic required, including NFA/DFA, and PDA. However, the editor in Turing Machine is still having malfunction. The platform should be easy manageable and configurable because the client, Dr. Mostafa Mohammed and other instructors and students, could use this software heavily. We need to make it as easy as possible. The UI part of the platform mostly is already designed and we added more buttons and features to make it useful. However, aesthetic is not heavily in our consideration since only the instructor would be dealing with our designed platform, whereas the student would be doing exercise on the OpenDSA platform. Data input is for the instructor to upload JFLAP file to the site and it will be converted to JSON file for the auto grading system. Then, the auto grading system takes the answers of the student to check the correctness in each case, and the result will be shown underneath the graph in a table. The result of the test would be stored into an object inside the exercise grader and shown

on the screen as a form of alert, containing attempts, test results, highest scores, and time consumed.

Requirements

Our project seeks to provide a complete framework to upload JFLAP exercises to the OpenDSA textbook for teachers to assign to students. This includes the processes to:

- Generate exercises

There needs to be a website to generate JSON files from JFLAP files, and functionality to create HTML pages to display the exercises. Exercises can include DFA's, CFA's, PDA's or Turing Machines, so framework must be built for each. Students must be able to input a suggested solution, which should then be tested against the appropriate conditions.

- Auto-grade exercises

Each exercise needs to present a point value for a student's input, involving checking the students response for errors and dividing credit appropriately. The grade should be displayed and the student should be able to redo the exercise, if allowed.

- Store students answers and grades in OpenDSA database

Grades for each exercise should be stored and accessible to the instructor. A structure to present more relevant grades for an exercise, such as more recent or higher points, should be available as well.

Design

Our program will seek to allow instructors to upload JFLAP files and use them to produce exercises in the OpenDSA textbook. This first involves giving the instructors a way to upload their JFLAP files, which contain the exercise. An algorithm should exist to convert the information in any JFLAP file into the appropriate form for a JSON file, as well as provide an option to download this JSON file for later reference. JFLAP files could describe DFA's, NFA's, PDA's or Turing Machines, and so must be able to recognize the type of exercise they are analyzing. This also means the JSON files produced will require different values depending on the type of exercise that they describe. Once the JSON file has been generated, this information should be utilized to generate an HTML page to display the exercise for students to view. Students should be able to input a solution for the instructor's algorithms to check, which should be reachable by the HTML page. The page should then produce a score, based on how well the student did, how close they were to the solution and how many test cases they were able to pass. This information should then be stored in JSON files for later reference from the instructor. The JSON files should also be able to update based on any more recent grades passed in as students try to receive a full score on the assignment. Each assignment should be able to store the amount of time it took for a student to complete that given assignment as well, so that the instructor can view whether a particular problem was more difficult or if students are quickly and randomly entering answers.

Implementation

createDFAexercise.html:

This file determines the layout of the exercise generator page. This page can divided into three parts: the radio button to select the type of machine need to generate, the description of the problem and its test cases. Finally, the button at the bottom to get the exercise.

createDFAexercise.js:

Determines the logic of the page and the function for each button. Its main functions can be divided into two kinds. One is to add new HTML statements to the above file. If the user wants to add multiple test cases or problems, this JS Script can adds new forms for the user to enter. The other is to store and process the data entered by the user. This script collects the data entered by the user and stores it in a variable. When the user need those data it can pass them to the user.

FAEditor.js:

This JS script has FA object which encapsulated the JSAV graph API. And some interface for the user to edit this object. Different webpage will have different buttons, and those buttons will connect to the API which provided by this JS script. These APIs include adding nodes and edges to the FA object, traversing the object, and so on. If you want to know all of them, please see the comments in the code.

ExerciseController.js:

This script determines the logic of the practice page. It works with the logic of the above script to complete the exercise page. When FAEditor.js creates the FA object, it determines which page called the current script. If the exercise page calls it, FAEditor calls ExerciseController.js and creates a new exercise object and stores it into a global variable called exerController. Creating the exercise object requires passing the path of the exercise

JSON file which generated by the first html page. So we can generate different exercise pages by passing different JSON file's path. In addition, The script provides function to check whether the answers provided by the students are correct or not. There is also a function to save all the attempts submitted by students into an JSON object.

serializableGraph.js:

This script mainly contains some functions that assist the user to manipulate the FA graph object. I mainly used the serialize() function which can convert the JFLAP file to a JSON file.

Testing

After each feature was implemented, we ensured to test that feature through the view of a user, and check that every aspect of the implementation was working appropriately. For the upload functionality, we made sure that the files were uploading correctly. Part of this check involved making sure that the files were also run through the appropriate formula to convert the files from a JFLAP format to a JSON format. Once the program had been extended to accept PDA's, we also had to test that the transformation was functional for this part of the execution. Examples for PDA's were sent through the program to double check whether or not the algorithms handled this design correctly, and that the output file adhered to the previously supplied HTML input acceptance. Turing Machines needed to be tested through providing such a machine to process, and then looked over after the code was finished running. The newly produced visual format was tested by observing how the new display appeared on the website for grading code.

User Manual

Professor:

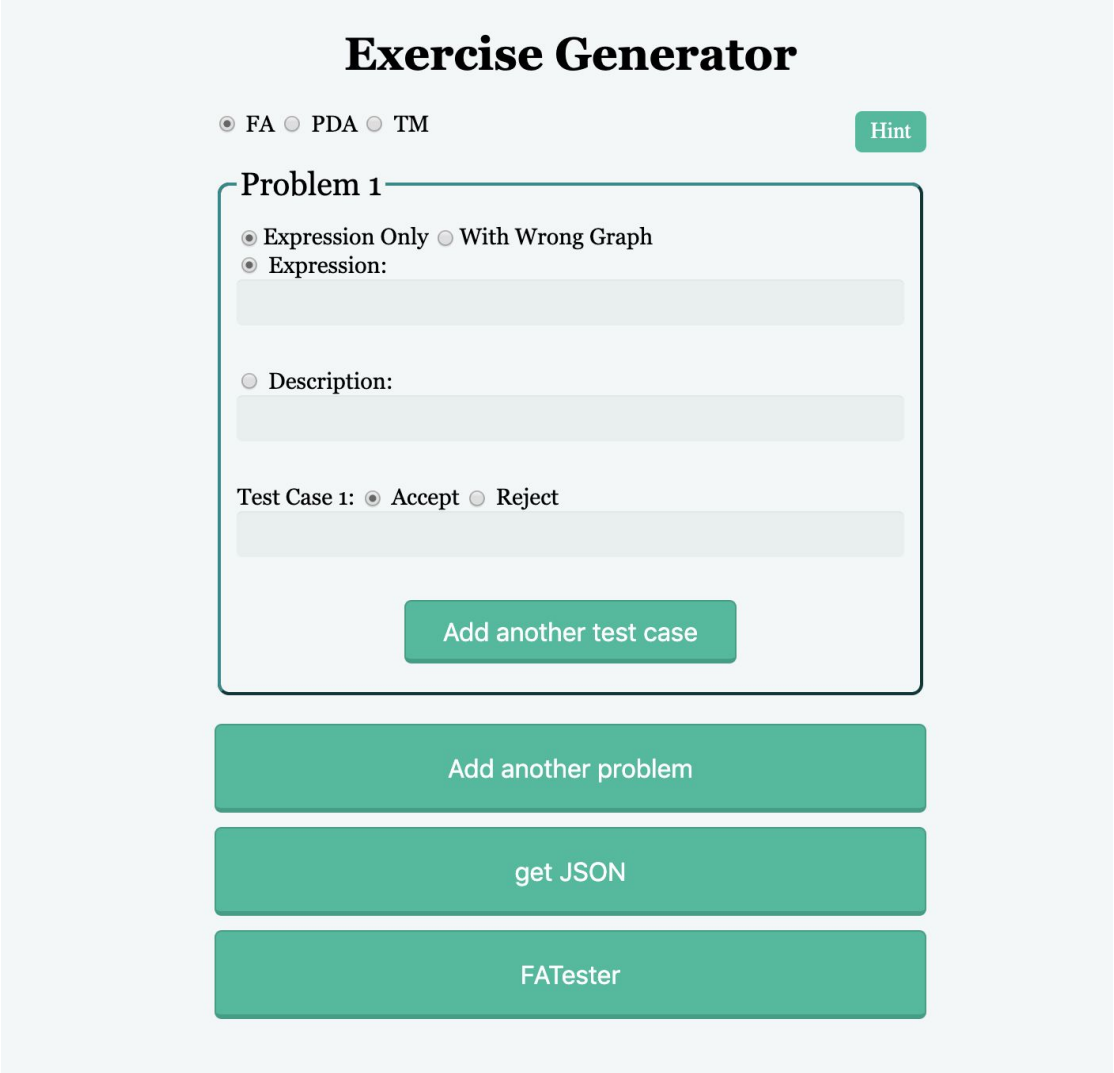

Figure 1 basic page

1. Load up Generation Website
2. Select Exercise Type
    a. FA: Finite Automata
    b. PDA: Pushdown Automata
    c. TM: Turing Machine

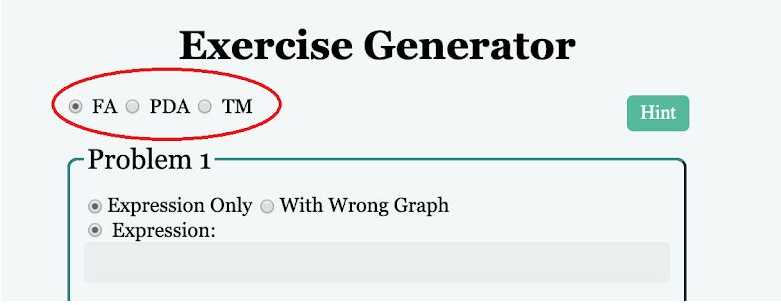

Figure 2- Machine selection

3. Select Exercise Format
    a. Expression Only Requires student to build entire graph for the exercise
    b. Wrong Graph will have the student either fix a graph or build off of a partially complete one.

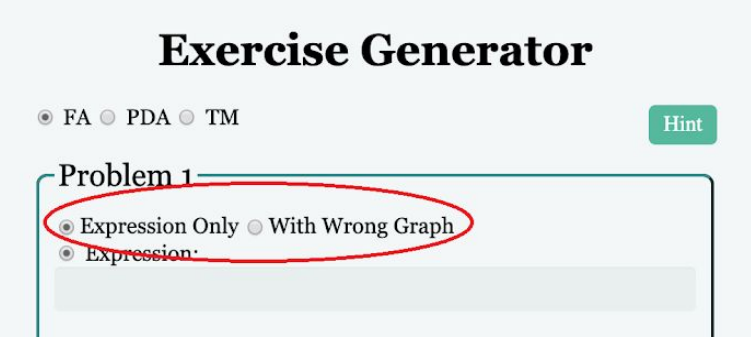

Figure 3 - Problem Type

4. Next provide the expression that the graph will represent.
5. Following that provide a description of what the student is supposed to do for the problem and any hints or other information that you want the student to have.

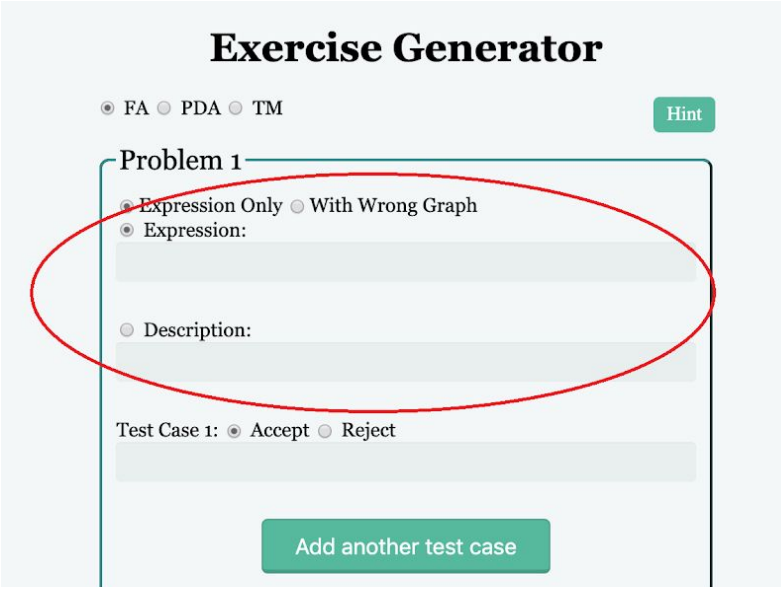

Figure 4 - Problem statement

6. Next is to specify the test cases that you would like to use for grading the graph. You can generate as many as you want and should specify the string and if that string should be accepted or rejected by the graph. These test cases will be used to grade the graphs and also to provide feedback to the students.

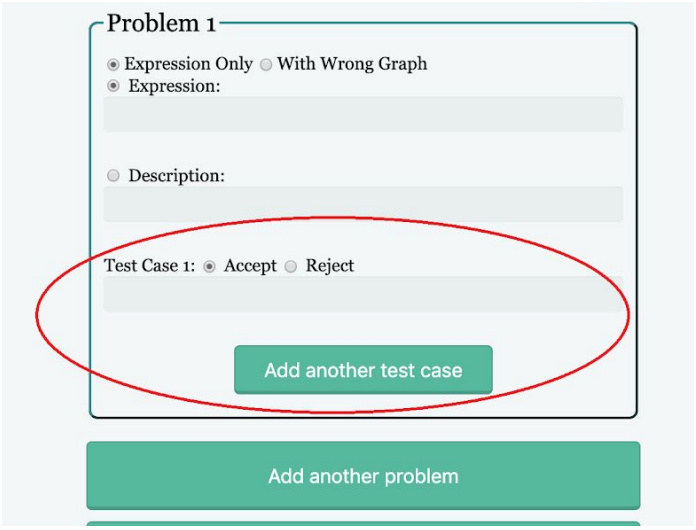

Figure 5 - Test Cases

7. Now to the graph itself. If Expression only was chosen then there is nothing else to be done. However if the wrong graph option was chosen then you need to specify a graph this can be done one of two ways.
    a. Create graph using provided tool
    b. Upload a graph built in JFLAP

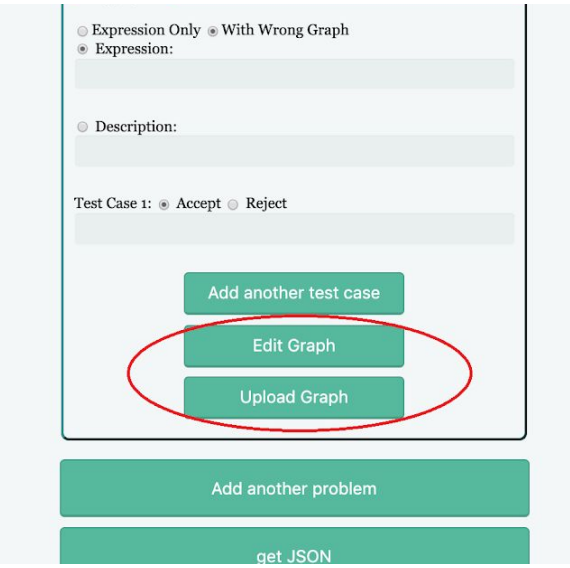

Figure 6 - Upload and edit graph

8. The edit graph button will take you to a page in which you can build the graph that you would like to be sent to the students along with the problem. The different graph editors are the three shown below.

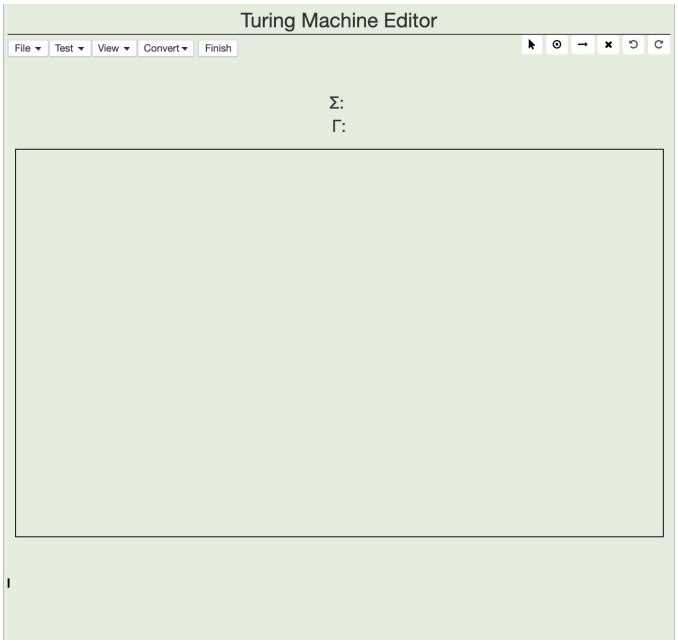

Figure 7- Turing Machine Editor

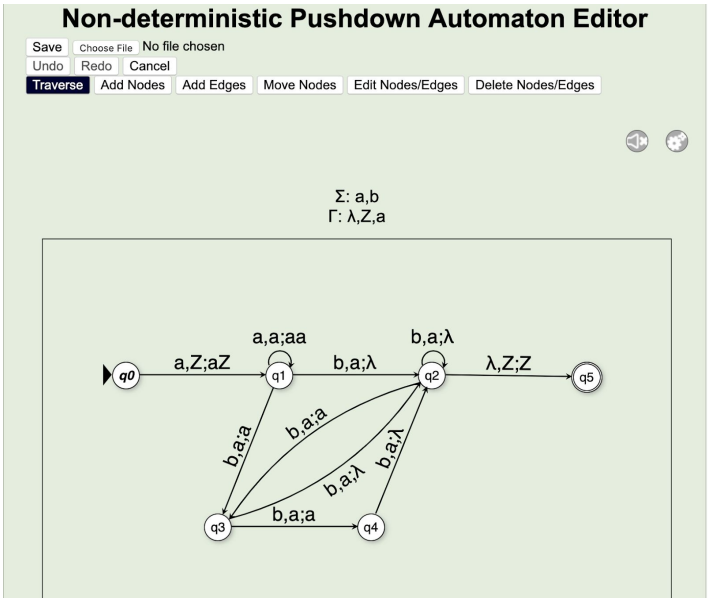

Figure 8 - PDA Editor

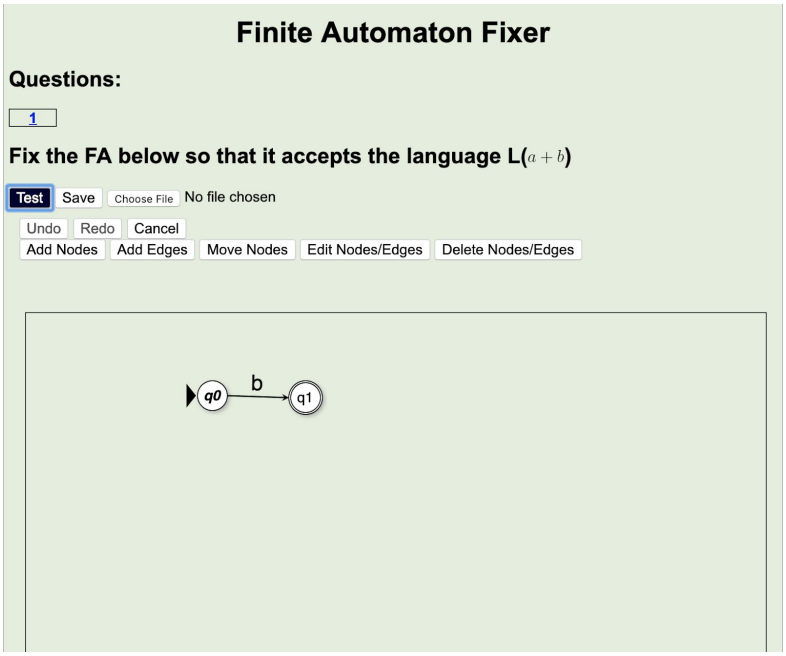

Figure 9 - FA Editor

9. Next you can decide if you would like to add another exercise to the problem set and if so repeat the process listed above.

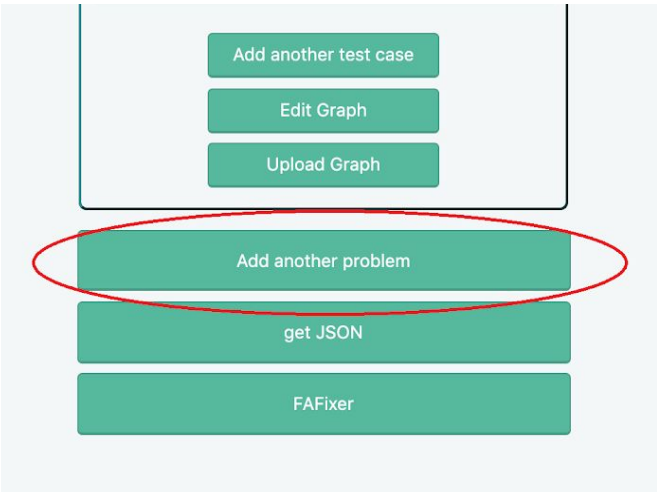

Figure 10 - Add additional problems

10. Now once all of your problems are completed you can now click the generate button to finish.

11. This button will create a folder containing an HTML page for your exercises along with the JSONS for each problem within the exercise set.

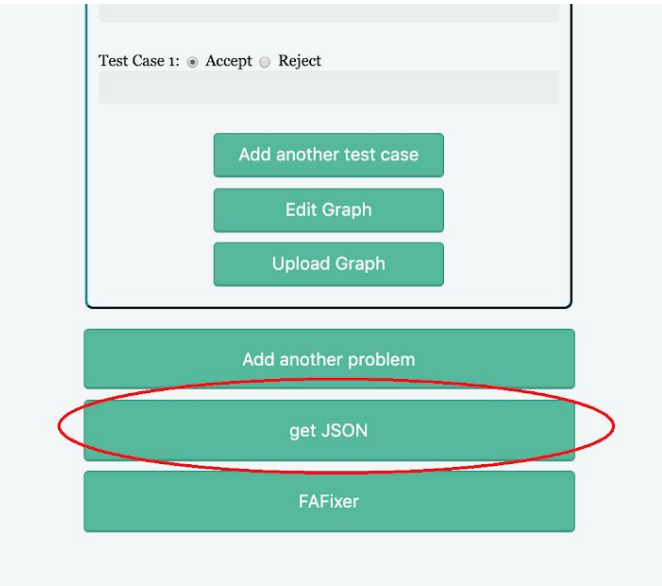

Figure 11 - get JSON

12. Download this folder and add it into the opendsa book where you would like.
13. When students work on exercises JSON files will be generated containing information about their attempts and score.
14. For each exercise the overall highest score will be recorded
15. For each problem the highest score, problem statement, test cases, and number of attempts will be scored.
16. For each attempt the score, which test cases passed, the time, and the submission will be stored along with the attempt.
17. These logs should be stored in the same folder as the exercises.

Student:

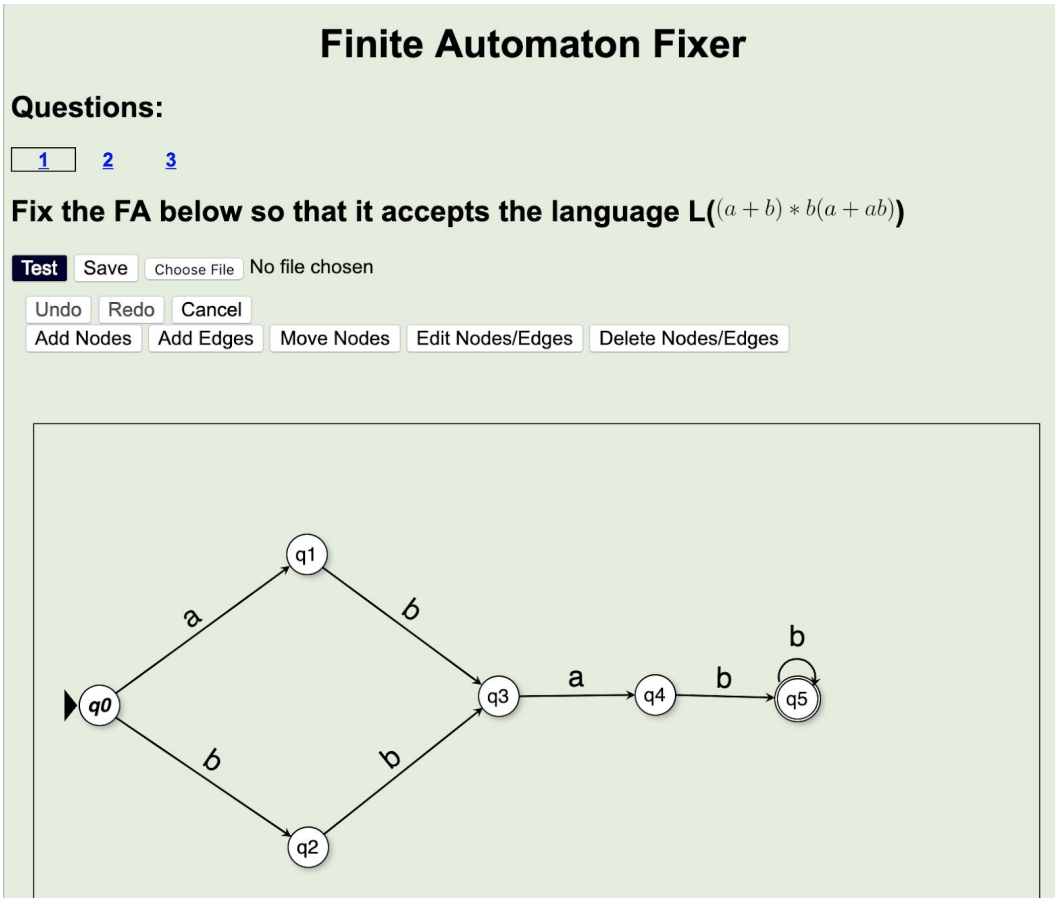

Figure 12 - Example of Editor

1. When you open the exercise you will either see a problem statement and either a blank area to draw a graph or a incomplete/incorrect graph.

2. You can use the provided buttons to add/remove states and edges to the graph and edit existing ones.
    a. To specify final and initial states you need to alternate click on the node you are wishing to change.

3. Once you are happy with the graph you can hit the test button to run your graph against the provided test cases.

4. A table will appear showing which test cases you passed and which ones you failed. If you are leaving the assignment or are switching to a different problem in the set you should use the save button to save your current graph so none of your progress is lost.

Correct cases: 3 / 5

Test Case	Standard Result	Your Result
bbbbab	Accept	Reject
baabbaa	Reject	Reject
aaabaa	Reject	Reject
bbab	Accept	Accept
abbab	Accept	Reject

Figure 13 - Test Case Feedback

5. You can use the blue numbers to navigate between the different problems in the exercise set.

6. At the moment, the design of accessing the log record of every attempts for each question is done by pressing the “test” button and an alert will pop up and display the results containing attempts, results, highest scores and time used.

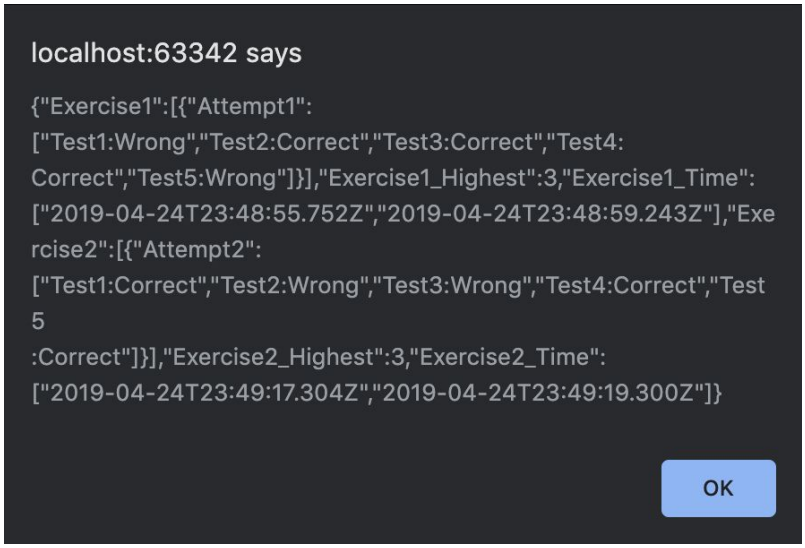

Figure 14 - JSON Log Format

Developer Manual

This project is to create a web page to help instructors to generate problems. There are two kinds of problems, one is FATester and the other is FAFixer. For FAFixer, the user needs to provide an expression, some test cases, and a wrong graph of finite automata. Students need to fix this wrong graph. FATester is very similar to FATester except that the exercise does not provide the wrong graph. Student draw the graph by themselves.

If student open the finite automation editor directly, the website won't show the "finish" button . This will prevents students from changing the question pool. Client can edit the graph by clicking the "edit graph" button in the exercise generator.

How is the exercise page generated?

The layout of these two exercise pages is determined by FAFixer.html and FATester.html. Editing the graph is done by FAEdior.js. Testing the graph is determined by exerciseContraller.js. When the user opens FATester.html or FAFixer.html, the browser will run the code inside FAEdior.js. First, they will judge whether the user is open those two exercise pages, if not, it will enter the logic of the general FA editor. Otherwise the exerController global variable inside the FAEditor.js will be initialized. To initialize this variable, we need to pass in the relative path of the JSON file corresponding to the exercise. There is a load equation in exerciseContraller.js that can read this JSON file into the browser. After that, the expression updateExercise is called to display the regular expression and the corresponding finite automaton on the web page. Clicking the Test button will call the startTesting equation. This equation will call the will reject equation in TraverseAccepter.js to determine if the image after the student's editing passes given test cases.

Upload function:

Because we found that users draw their own pictures is a very time-consuming thing. So we added a button to upload an image on the page. The process of uploading the function is shown below:

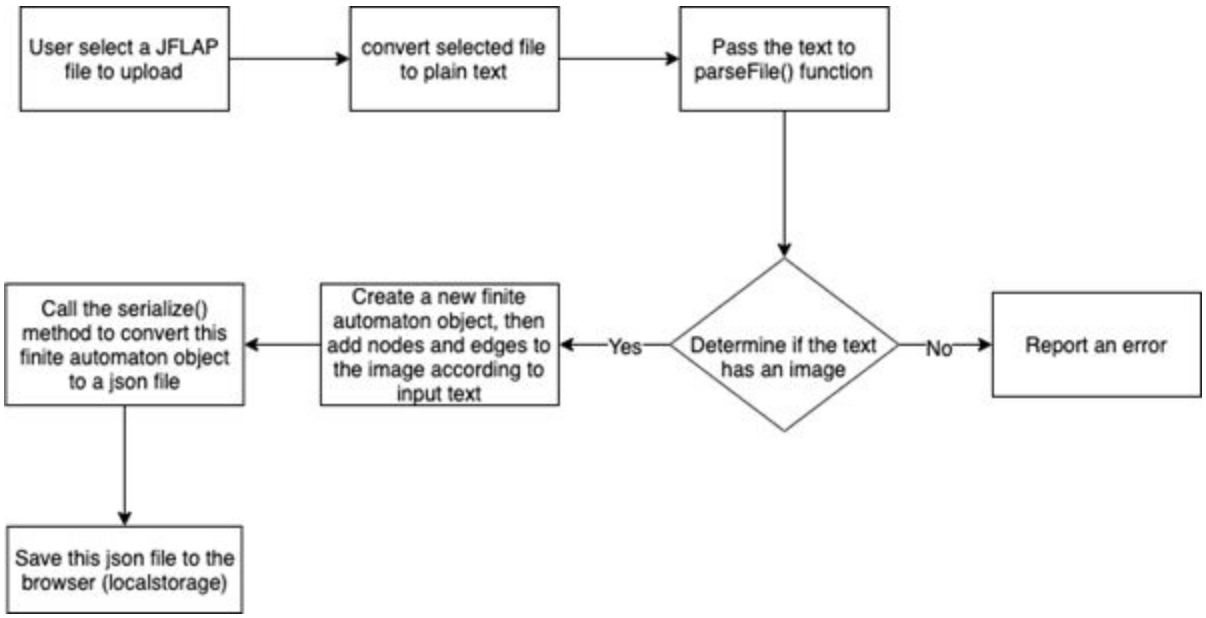

Figure 15 - Process Map

FAFixer and FATester Button:

In the process of development, we found that after the user created the problem, they did not know what the problem looks like. They could only download the corresponding JSON file. So, we added two buttons to help the user test their work. When the user clicks one of those two buttons after completing the question. The web page will integrate all problem descriptions and graphs into a JSON file and save it to the current folder which HTML page locates. User can change the address of where to load and save the file from the JavaScript.sup

Collect student grade data:

During the development, we were required to complete a log, containing the information including test results, highest score, attempts, and time spent. We obtained the information from the JavaScript file from the exercise controller and saved the data to an object as a log in the global variable. Users could use and change the content and structure of the object relating to their needs.

Lessons Learned

Timeline

Table 1 - Milestones

2/7 - Getting Started	● Go over and finalize requirements ● Set up GitHub ● All members have access to edit source code ● Download and install Python
2/28 - First Steps	● Add a button to upload & convert JFLAP file to JSON ● Set up foundation and framework on site for later iterations
3/18 - Finishing Generation	● Upload and generate HTML/JSON for exercises ● Add mechanism to allow instructor from generating exercises by reading predefined JSON files (generated from website)
3/25 - Expanding Functionality	● Build structure for JSON files detailing PDA's and Turing Machines ● Allow instructor to generate exercises for models other than DFA's/NFA's like PDA's ● Develop web-pages for each new model (as described above) ● Fix exercise evaluation display
4/8 - Exercise Grading	● Grade student work ● Respond to incorrect answers appropriately ● Allow instructor to generate exercises for Turing Machine models
4/15 -	● Storing of student work and grades ● Grades are accessible ● Tested for errors in calculations
5/1 - Finishing Touches	● Generate submission log for JSON format ● Incorporate updated editors ● Only important grades are recorded ● Website is ready to be used
Bonus	● Private Finish Button

Problems

During the process of implementing the function of generating and grading exercise, we encountered some bugs that already existed in the base code. For example, there is a function within the FAEditor.js file, called ParseFile. It is used to convert JFLAG file into an FA graph. However, there is a global variable in the function that caused the method to not be able to be used in other files. Therefore, we added the function into the createDFAexercise.js file as well. There was also a problem with the testing method, that would return whether the graph drawn is passing all the tests. At first, it did not return the right testing result, which was caused by the boolean method called willReject, but later it was fixed.

The current practice page can only be used to read the JSON file needed to generate the exercise from a fixed path (localStorage). Our next goal is to make the practice page read any JSON file in the same directory. In this case, the client can add new exercises by uploading the new JSON file to the specified folder. Our client then can invoke the HTML inside any website.

In addition, we need to connect the practice page with the database of course grades so that the teacher can score assignments.

As the process continued, we also found that different priorities became more important. Grading assignments became less important compared to completing the process of uploading and processing JFLAP files. Therefore, we wanted to work towards developing PDA's and Turing Machines before completing the tasks related to storing and grading students' progress. This became in conflict with our milestones, which originally would have had grading implementation started in March.

Solutions

To solve the current problem, we will move the HTML file into a new created directory. Instead of reading JSON files from local storage, it will read files from the same directory of HTML. The “Get JSON” button will not only download the JSON file but also store the file in the new directory. We will update all the JavaScripts’ file paths included in the HTML to ensure every js files will be found correctly.

In order to prioritize working on the JFLAP files as a whole before implementing the grading structure, we shifted our schedule to match this goal. Coding for uploading and editing PDA’s and Turing Machines was moved to March, along with creating conversion from JFLAP into JSON. Additionally, the goals towards grading solutions was moved into April, giving us the time needed to work on this feature.

Future Work

There are still some bugs within the Turing Machine. We will need to address these problems to make the user experience better and the editor functional. We will follow the instruction from our client and deliver in time before the semester is over.

Within the next week, we will also create a button for the user to view the testing result whenever they want to, instead of pressing the “Test” button. This function should be very easy to implement and achieve.

Acknowledgements

Client:
Professor Mostafa Mohammed
profmdn@vt.edu

Platform:
OpenDSA Interactive Textbook
http://lti.cs.vt.edu/FL/Books/VisFormalLang/html/

References

OpenDSA Interactive Textbook - Formal Languages With Visualizations
Created as part of the OpenDSA hypertext project
http://lti.cs.vt.edu/FL/Books/VisFormalLang/html/

This is the online textbook that our program will be running through, and where instructors will be able to upload exercises to. This resource also provides explanation for the various exercises that can be uploaded and provides explanations as to how these structures work. Knowledge of the workings for these assignments is invaluable to our project, since we need to make sure we implement each assignment appropriately.