



CS6001 Formal Specification & Software Implementation

Coursework 1: Individual Report

Finite State Automata

By: Lauren Spruce

ID: 18011848

Table of Contents

CS6001 Formal Specification & Software Implementation	1
Coursework 1: Individual Report	1
Finite State Automata	1
Table of Figures	2
Introduction	3
Class Diagram NFA Sequential Search	4
Sequence Diagram	5
Description of the Program	6
Test Data	7
Running the program	10
Output produced	12
User inputs	12
Using a Text File	13
Conclusion	14
Source Code	15
SequentialSearchFilter Class	15
SequentialSearchNFA Class	16

Table of Figures

Figure 1: Transition Diagram for NFA, which recognises the language {a,b}	3
Figure 2: Class diagram for NFA program	4
Figure 3: Sequence Diagram for the program	5
Figure 4: NFA: Configuration Sequence Diagram for abababaa	7
Figure 5: NFA: Configuration Sequence Diagram for bbabb	8
Figure 6: NFA: Configuration Sequence Diagram for aaaaaa	8
Figure 7: NFA: Configuration Sequence Diagram for bbbbbb	9
Figure 8: NFA: Configuration Sequence Diagram for aabbbb	9
Figure 9: NFA: Configuration Sequence Diagram for babbbb	9
Figure 10: Setting the to find the java executable	10
Figure 11: Changing the path to where the java files are located	10
Figure 12: Compiling the Java files	10
Figure 13: Java files compiled	11
Figure 14: Running the program	11
Figure 15: Showing where the java files are copied to with the packages removed	11
Figure 16: Screenshot of the user inputted words within language {a,b}	12
Figure 17: WordsNFA.txt	13
Figure 18: Using the WordsNFA.txt file to output results from the program	13

Introduction

Continuing from our group coursework, we were assigned to each implement a version of the automaton. This was the NFA before and after the conversion to a DFA. I implemented the NFA using sequential searching. The other version was a parallel search, followed by the DFA implementations using direct encoding and data structure initialisation.

The choices of implementations were all agreed by every team member during the meetings held for the group report and the creation of the NFA. In this report I will be showing the two UML diagrams I have created, these being the UML class diagram and the Sequence diagram of the program. I will also be describing my program and showing my test data I used to test the program. This will be followed by screenshots of the program running and the outputs produced by my test data. Finally, I will include all the source code within my program.

Below shows our NFA transition diagram we implemented during the group report. This is what I modelled for my program. This NFA recognises the language $\{a,b\}$ the constraints of our NFA are as follows:

- The NFA can accept only a.
- The NFA only accept if a is part of the last three characters.
- The length of the words can be less or bigger than three.

We start by building a start state with a loop a,b that will allow us to build any word over $\{a,b\}$. Next, we need to take care of the NFA last three characters and make sure a is part of it. Once we have the list of accepted ending, the next step will be to create next states so we already the start state with a loop a,b now we will states that correspond to the language where the first of the last 3 characters start with a and where the first of the last 3 characters start b. Then we can reduce the NFA by asking ourselves if two states can be combined into 1. This is the result of the NFA with less states as possible.

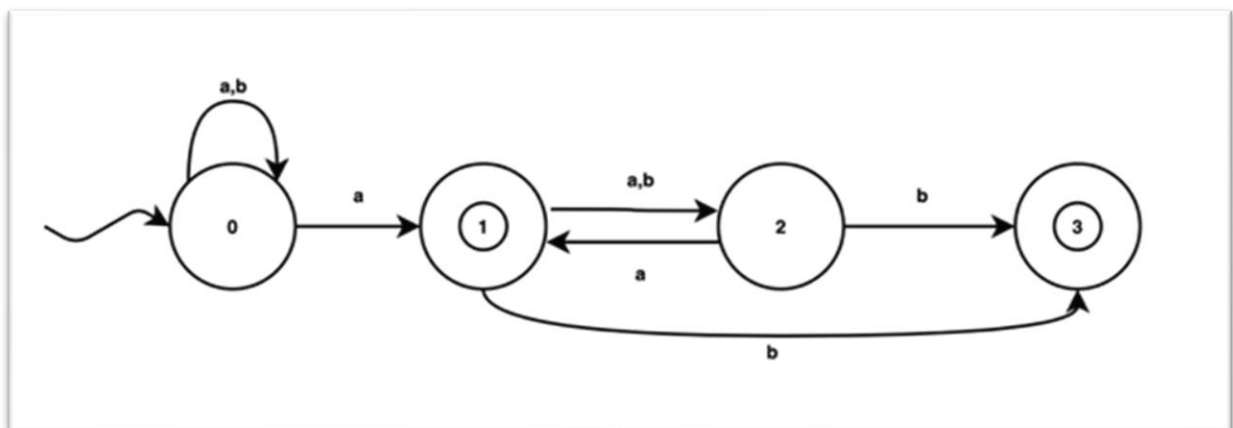


Figure 1: Transition Diagram for NFA, which recognises the language $\{a,b\}$

Class Diagram NFA Sequential Search

First, before implementing the NFA into code, I created a class diagram to ensure I had a plan and model to refer to when coding. This was to ensure my program was maintained and the end goal was never altered during my development process. It allowed me to translate the model easily and efficiently into suitable code.

The 3 main classes on this diagram consist of the Filter, the Sequential Search, and the characters. The filter class is to ensure that the strings of words are read using the Buffered reader. It allows for text and text files to be read. It starts to read the lines as they are inputted, following this each string will get printed out to the terminal, with the following accepted or rejected string.

The Sequential Search class is where I create each state in the transition diagram and program which states that accept a or b. This is done by using a 3-dimensional array. I am explaining this in more detail in my description of the program. Finally, the characters class was to refer to the strings, the new method shows a new string being inputted in the program and the read line is to make sure the program reads the next lines when reading the text inputted by a user or from a text file.

The filter and the Sequential search have a 1 to 1 relationship as the program is only reading one string at a time. The character class is related to the filter as new and read line are methods performed within the filter.

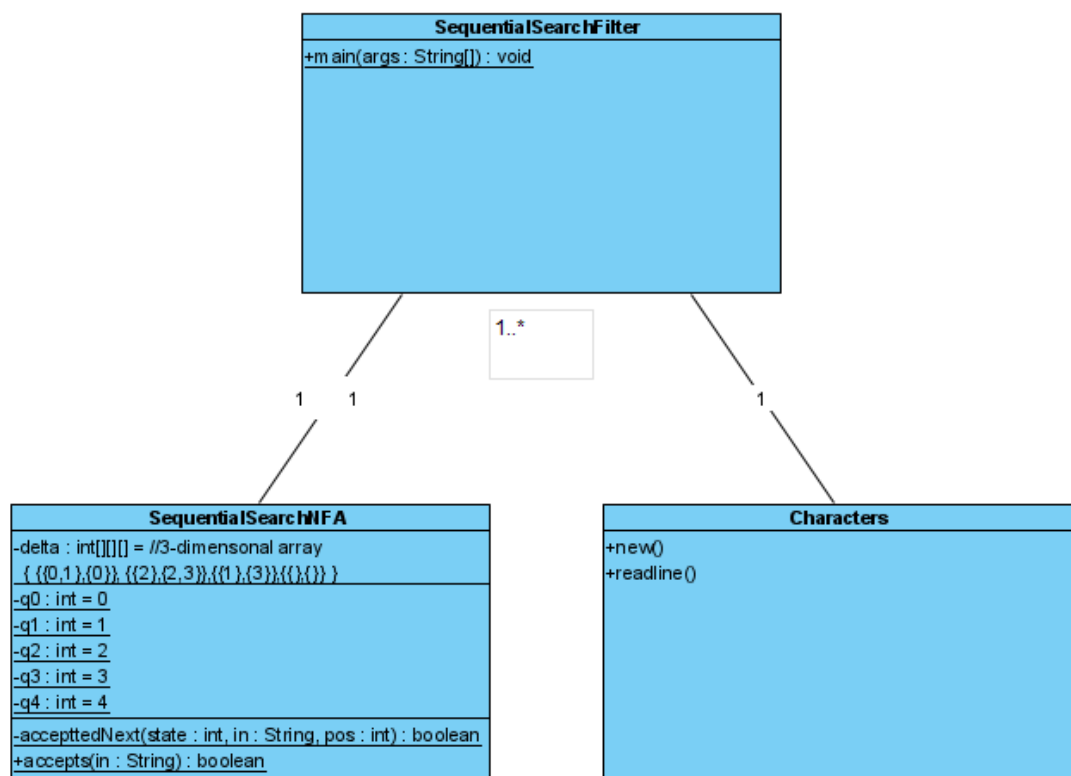


Figure 2: Class diagram for NFA program

Sequence Diagram

I will now be showing my sequence diagram. I used a sequence diagram as it represents the interaction the program has between all the classes. It also shows how all the methods work together and how overall functionality of the program behaves.

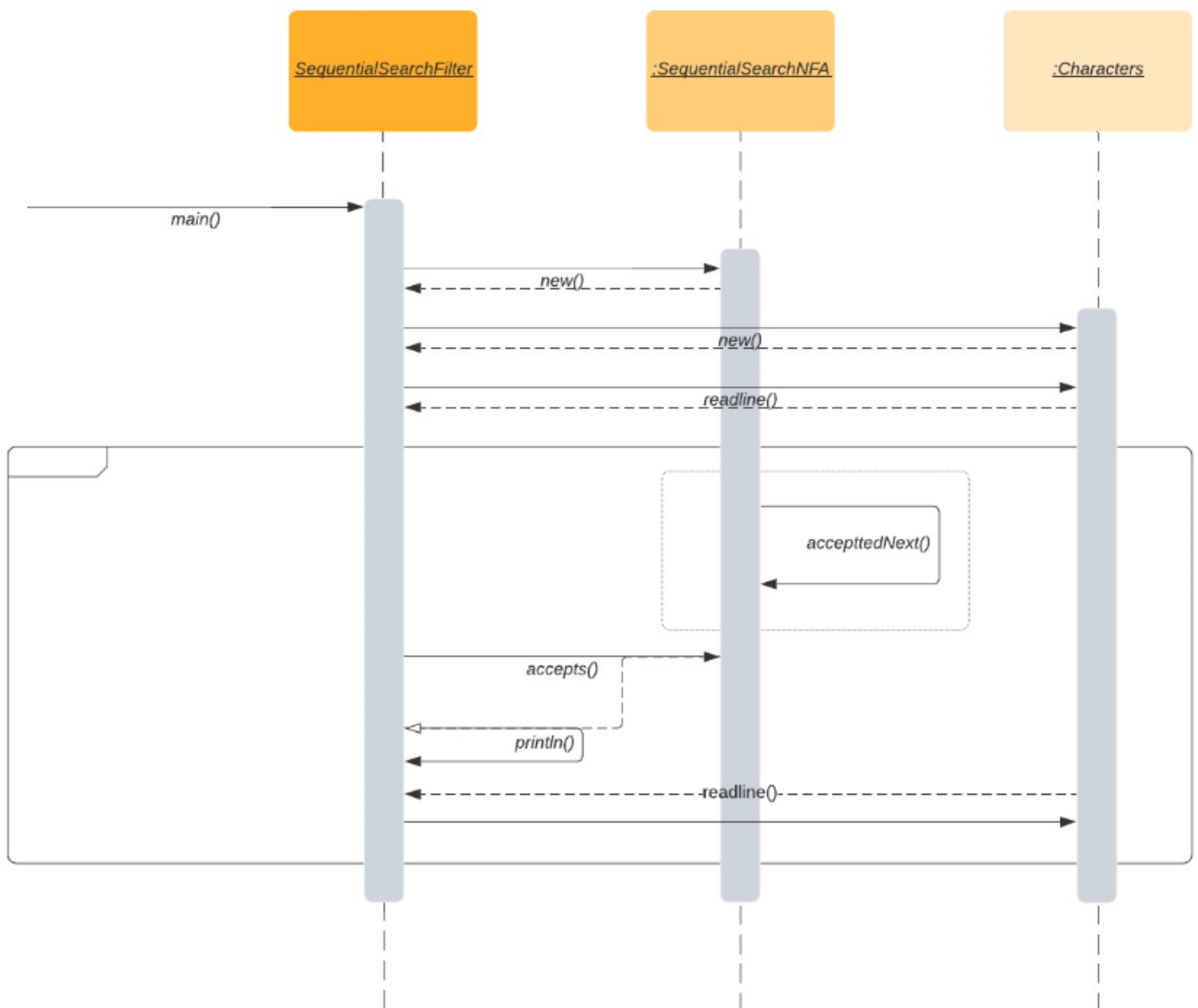


Figure 3: Sequence Diagram for the program

Description of the Program

My overall program consists of 2 classes. These classes are the Sequential Search filter and the Sequential Search NFA.

The 'Sequential Search filter' class filters any strings which are inputted by a user or a text file. For each line of words which are inputted, these will be accepted or rejected, then outputted to the standard output being the terminal. First of all, I outputted a few strings to the user to describe what the program's purpose was, in this scenario it is for a sequential search of my groups NFA, followed by telling the user to input a word from our language, {a,b}. I then use a buffered reader, which can read the file or user input. I created a new buffered reader and an input stream reader to allow text and files to be read from line to line. I then created a string s, this string was what the user inputted and it starts to read all inputs. By creating a while loop, which is if the string is not empty, I put an if statement inside which pulls the method 'accepts' from the 'SequentialSearchNFA' class, this is to show all strings which are accepted by the NFA to the user. This output 'accepted,' is also followed by the string which the user inputted.

The else of my if statement will just show any strings which have not been accepted by the NFA with the following output message 'rejected' followed by the string which the user inputted.

The next class, 'SequentialSearchNFA' is where I created a three-dimensional array to store the transition functions of the NFA. In here, we are encoding the states because of the transitions. If the start state's input is a, this will equal [0][0][0] / [0][0][1] which represent the state we can go to {0,1} because if we input, we can go to state 0 or 1. If the start states input is b, this will equal [0][1][0] as b will loop back into 0.

I then included all the states from 0 to 4, I created state 4 to act as a sink state, this is for when we are in state 3. In state 3 the inputs a or b will not go anywhere so therefore the sink will allow them to be looped and stay in there.

I then created the 'acceptedNext' method this method is to accept any more strings. I created an if statement to make sure if there are no more a's or b's to read then it will return to state 1 or 3, this is because these are the final states in our NFA. Then I made a try and catch for the characters which are inputted, this was to advance the character to the next position using the difference in ASCII code. After the transition is done, we use the 'nextStates' array, this contains 0 or more next states. Each move is tried recursively, if it leads to an accepting state then the string is returned true. Finally, it will be returned false if all the moves of the character fail, rejecting the entire string.

Finally, I created the 'accepts' method which will accept all strings from 'acceptedNext' method, if the start of the recursion is in the state 0 or 3, being our final states of the NFA.

Test Data

The test data I have included in my program is 3 accepted words and 3 rejected words from our language, {a,b}. The same words were also used throughout our group project as we needed to make sure the words accepted by the NFA would also be accepted by the DFA we created and minimised. These words are as follows:

1. abababaa
2. bbabb
3. aaaaaa
4. bbbbb
5. aabbb
6. babbb

Below I have shown the accepted and rejected words, followed by their configuration sequence and diagram to prove they are either accepted or rejected from the NFA. These words are what I will be using to test my program.

Accepted words:

1. abababaa

Configuration Sequence and diagram:

$0 \vdash \text{bababaa} \quad 0 \vdash \text{ababaa} \quad 0 \vdash \text{babaa} \quad 0 \vdash \text{abaa} \quad 0 \vdash \text{baa} \quad 0 \vdash \text{aa} \quad 0 \vdash \text{a} \vdash 1 \in F = \text{Accepted}$

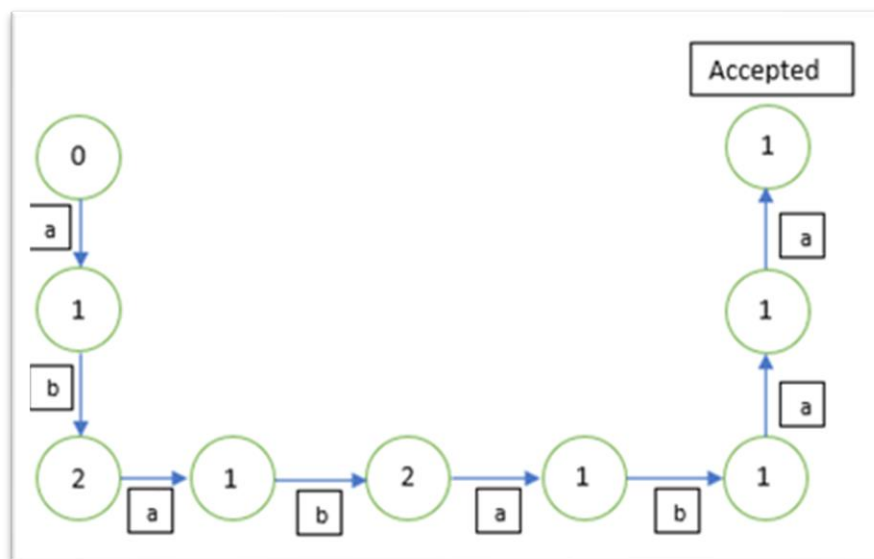


Figure 4: NFA: Configuration Sequence Diagram for abababaa

2. bbabb

Configuration Sequence and diagram:

$0babb \vdash 0abb \vdash 1bb \vdash 2b \vdash 3 \in F = \text{Accepted}$

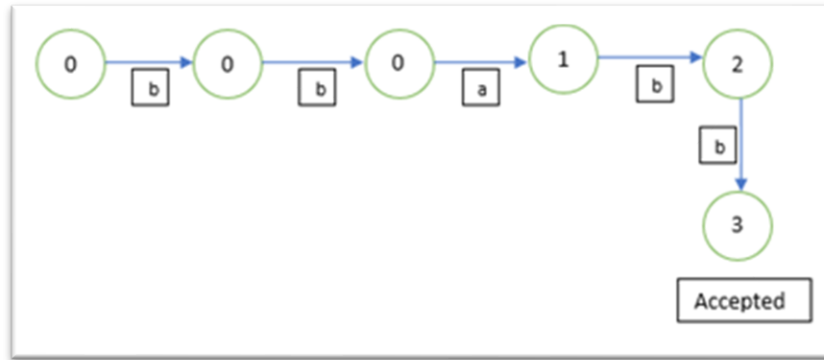


Figure 5: NFA: Configuration Sequence Diagram for bbabb

3. aaaaaa

Configuration Sequence and diagram:

$0aaaaaa \vdash 1aaaaa \vdash 2aaaa \vdash 1aaa \vdash 2aa \vdash 1a \vdash 1 \in F = \text{Accepted}$

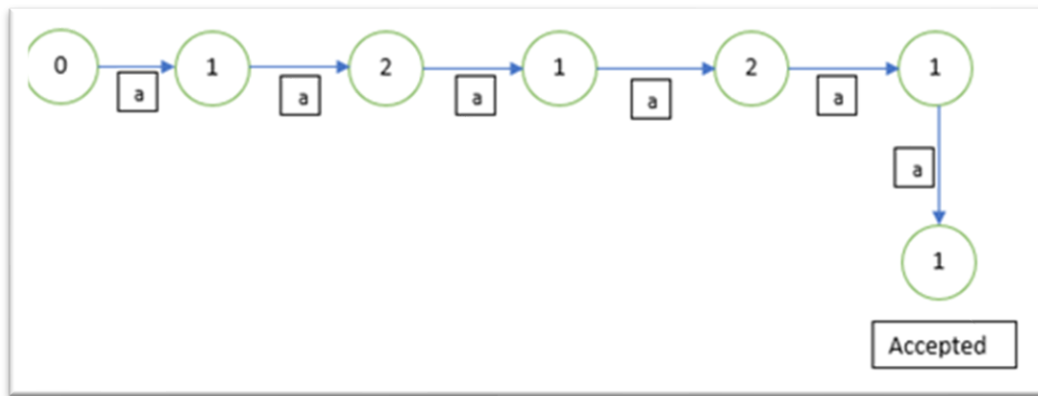


Figure 6: NFA: Configuration Sequence Diagram for aaaaaa

Refused Words:

1. bbbbb

Configuration Sequence and diagram:

$0\text{b}\text{b}\text{b}\text{b}\text{b} \vdash 0\text{b}\text{b}\text{b}\text{b} \vdash 0\text{b}\text{b}\text{b} \vdash 0\text{b}\text{b} \vdash 0\text{b} \vdash 0 \notin = \text{Rejected}$



Figure 7: NFA: Configuration Sequence Diagram for bbbbb

2. aabbb

Configuration Sequence and diagram:

$0\text{a}\text{a}\text{b}\text{b}\text{b} \vdash 0\text{a}\text{b}\text{b}\text{b} \vdash 0\text{b}\text{b}\text{b} \vdash 0\text{b}\text{b} \vdash 0\text{b} \vdash 0 \notin = \text{Rejected}$



Figure 8: NFA: Configuration Sequence Diagram for aabbb

3. babbb

Configuration Sequence and diagram:

$0\text{b}\text{a}\text{b}\text{b}\text{b} \vdash 0\text{a}\text{b}\text{b}\text{b} \vdash 0\text{b}\text{b}\text{b} \vdash 0\text{b}\text{b} \vdash 0\text{b} \vdash 0 \notin = \text{Rejected}$

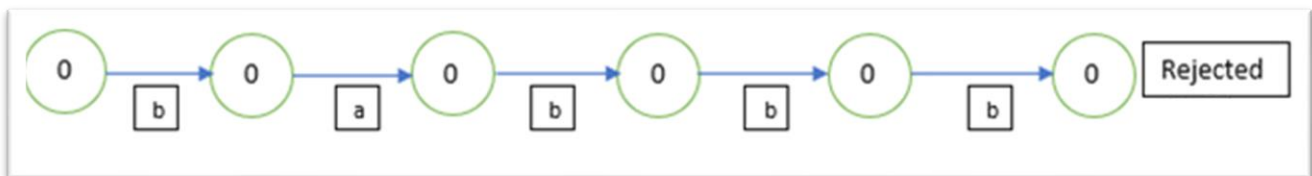
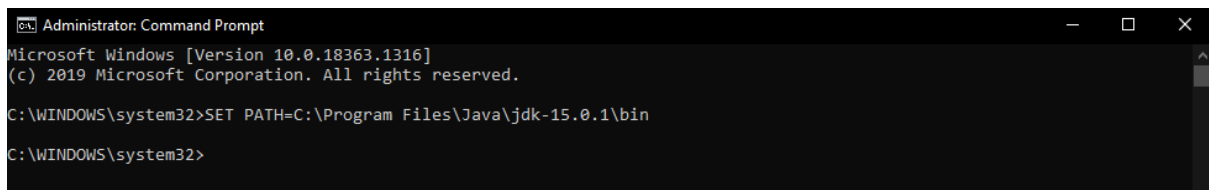


Figure 9: NFA: Configuration Sequence Diagram for babbb

Running the program

How to run the program. The program can be run on NetBeans or through the command prompt. To run the program on NetBeans you must right click on the project and select run. This will run the program on the command line within NetBeans.

For my demonstration purposes I will be running my program through the command prompt. I did this firstly by running command prompt as administrator. Once the command prompt was open, you must set the path to find the java executables, as shown in the screenshot below.



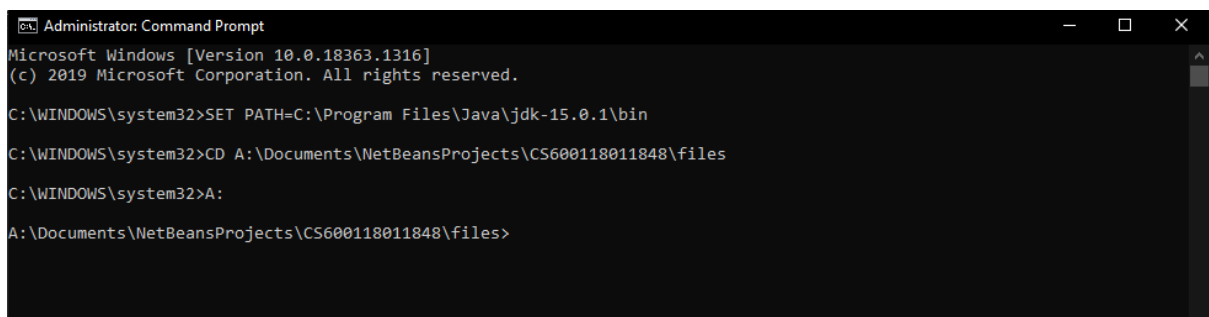
```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.18363.1316]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>SET PATH=C:\Program Files\Java\jdk-15.0.1\bin

C:\WINDOWS\system32>
```

Figure 10: Setting the to find the java executable.

Then, you must change the path to find where the java files are located within my project. You must also make sure you are selecting the correct drive, I had to change my drive to 'A:' which is what I named my HDD.



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.18363.1316]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>SET PATH=C:\Program Files\Java\jdk-15.0.1\bin

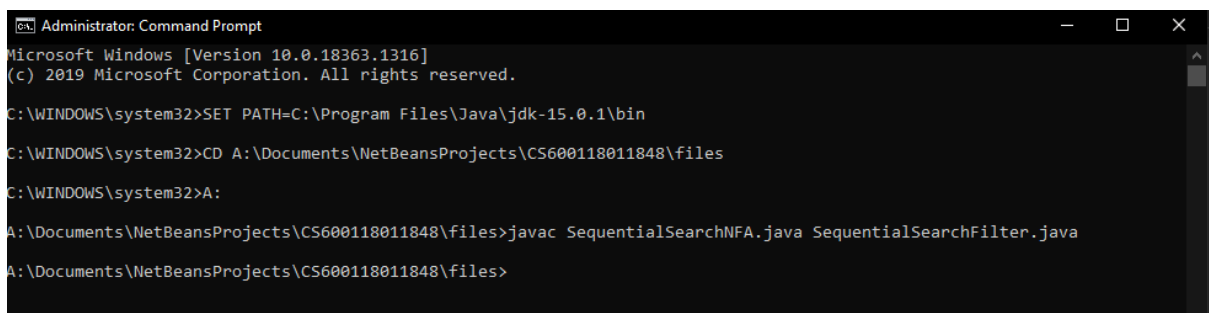
C:\WINDOWS\system32>CD A:\Documents\NetBeansProjects\CS600118011848\files

C:\WINDOWS\system32>A:

A:\Documents\NetBeansProjects\CS600118011848\files>
```

Figure 11: Changing the path to where the java files are located.

Then, you must compile the java files using 'javac', this compiles the file, once the files have been correctly compiled this will create class files, I show this in the screenshot below.



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.18363.1316]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>SET PATH=C:\Program Files\Java\jdk-15.0.1\bin

C:\WINDOWS\system32>CD A:\Documents\NetBeansProjects\CS600118011848\files

C:\WINDOWS\system32>A:

A:\Documents\NetBeansProjects\CS600118011848\files>javac SequentialSearchNFA.java SequentialSearchFilter.java

A:\Documents\NetBeansProjects\CS600118011848\files>
```

Figure 12: Compiling the Java files

This screenshot is to show the class files which have been created by compiling the original java files.

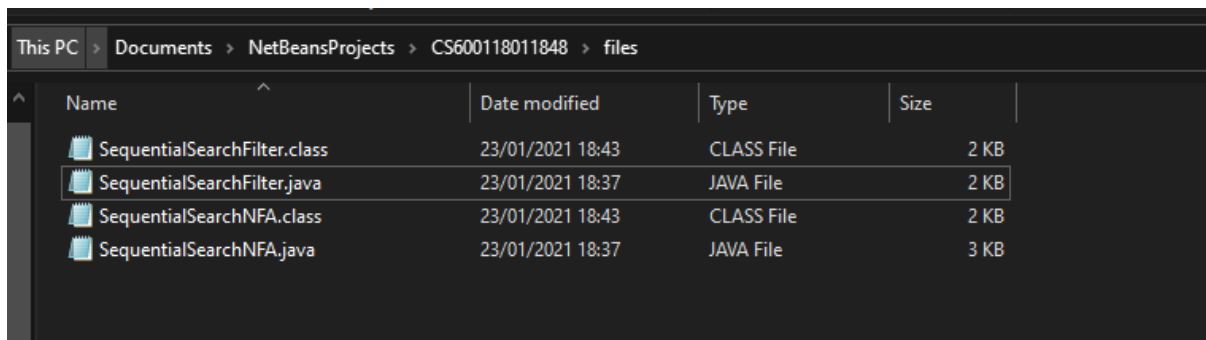


Figure 13: Java files compiled.

After compiling has been completed, you must type 'java filename.' This will then run the program through the command prompt, as shown below.

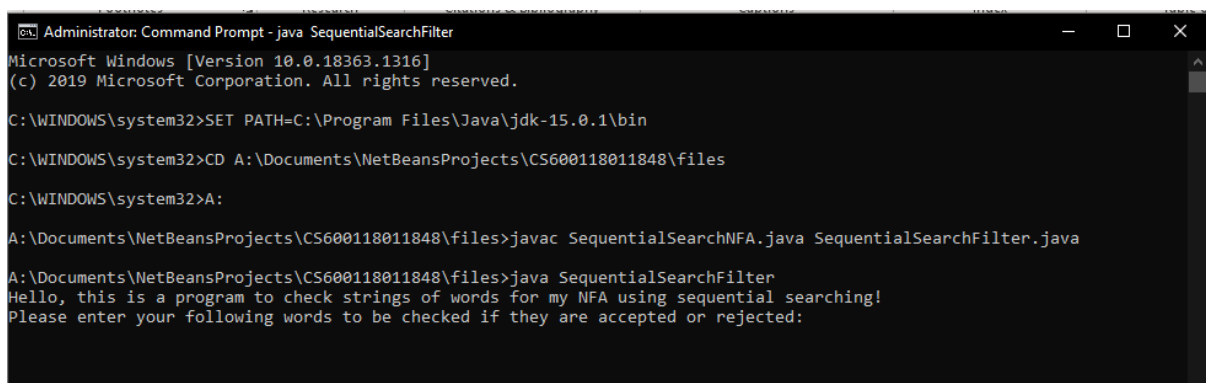


Figure 14: Running the program.

When deploying my program, I put the original java files into a new file within the project, I did this to remove the packages as NetBeans's created these. These packages make it so the program is unable to run through the command prompt, therefore this file is where I will be running the program from.

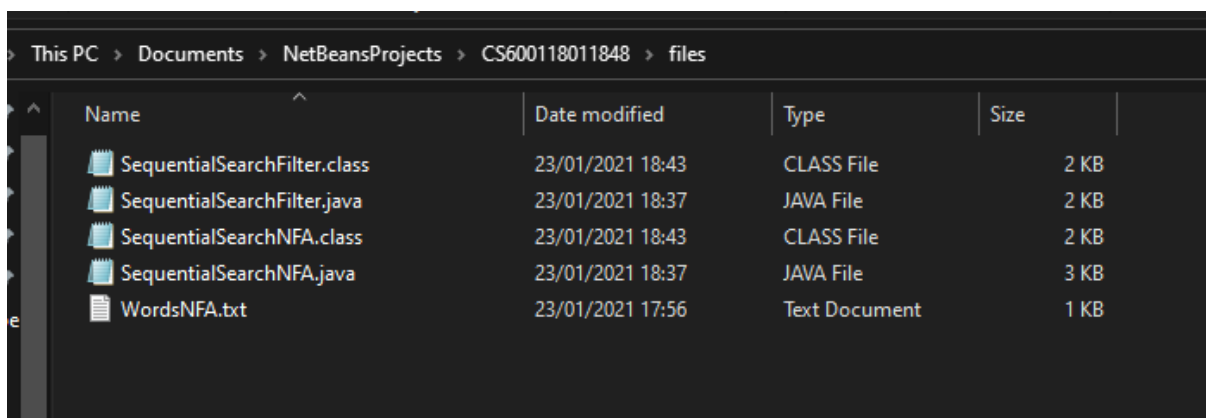


Figure 15: Showing where the java files are copied to with the packages removed.

Output produced

User inputs

After the program has been successfully run on the command line, the user can input words into the command prompt, this will then cause the program to return the strings, showing if they have been accepted or rejected. Within the screenshot below, I show that the output's produced correspond with the words that we used to check our original NFA. These words were as follows:

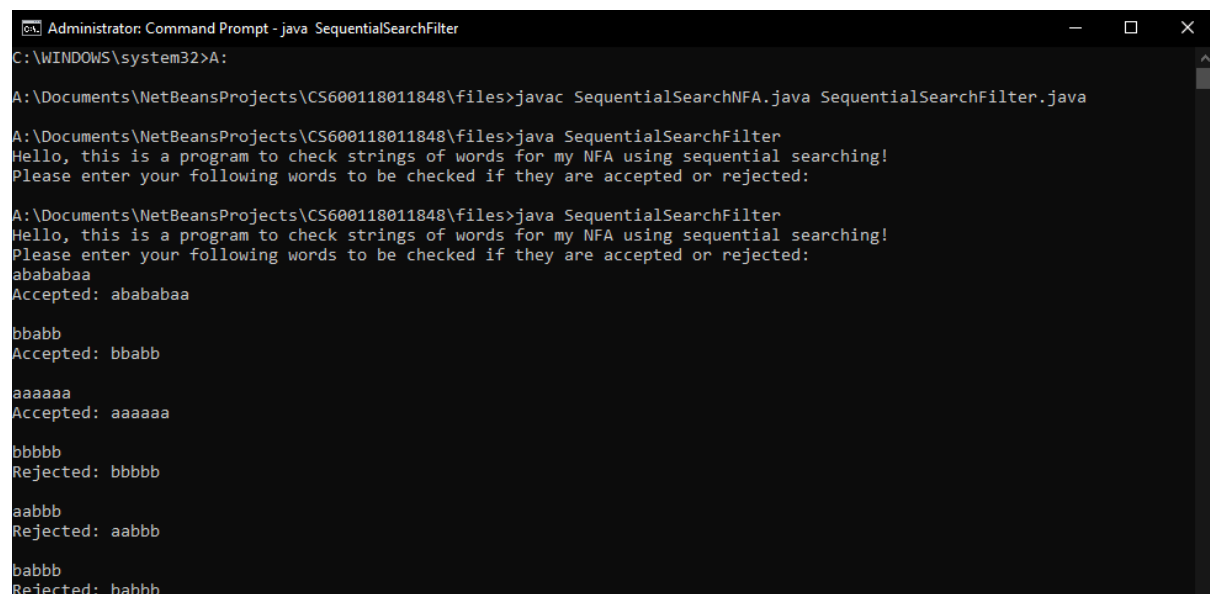
Accepted:

1. abababaa
2. bbabb
3. aaaaaa

Rejected:

1. bbbbb
2. aabbb
3. babbb

As shown below, all the corresponding words are correct within my program, so therefore my program has successfully implemented the NFA using sequential searching.



```
Administrator: Command Prompt - java SequentialSearchFilter
C:\WINDOWS\system32>A:
A:\Documents\NetBeansProjects\CS600118011848\files>javac SequentialSearchNFA.java SequentialSearchFilter.java
A:\Documents\NetBeansProjects\CS600118011848\files>java SequentialSearchFilter
Hello, this is a program to check strings of words for my NFA using sequential searching!
Please enter your following words to be checked if they are accepted or rejected:
A:\Documents\NetBeansProjects\CS600118011848\files>java SequentialSearchFilter
Hello, this is a program to check strings of words for my NFA using sequential searching!
Please enter your following words to be checked if they are accepted or rejected:
abababaa
Accepted: abababaa
bbabb
Accepted: bbabb
aaaaaa
Accepted: aaaaaa
bbbbbb
Rejected: bbbbb
aabbb
Rejected: aabbb
babbb
Rejected: babbb
```

Figure 16: Screenshot of the user inputted words within language $\{a,b\}$

Using a Text File

Another way the NFA can read inputs is by using a text file containing the words we used to test our original NFA. I created a text file with each word on a separate line, this is to ensure that each word will get read separately by my program and output the correct 'accepted' or 'rejected' message to the user. This text file was named 'WordsNFA.txt,' and is shown in the below screenshot.

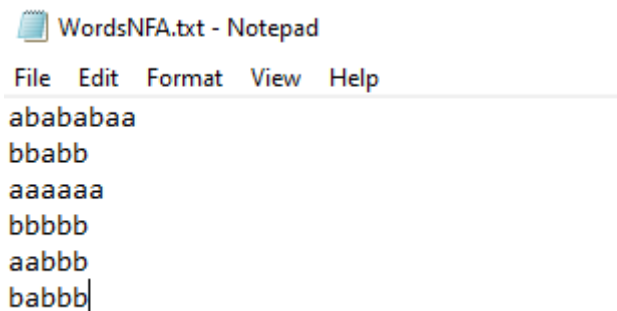


Figure 17: WordsNFA.txt

I then ran the program again using the command prompt, copied the file address into the command prompt followed by '< WordsNFA.txt.' This file is also located where the java and class files for my program are, so therefore the program can read all the words located within it. As the output produced is the same as the user inputted, as I used the exact same words, we used for testing our original DFA, it shows that the outputs are the same. Therefore, it is possible for my program to read either user inputted words or text files.

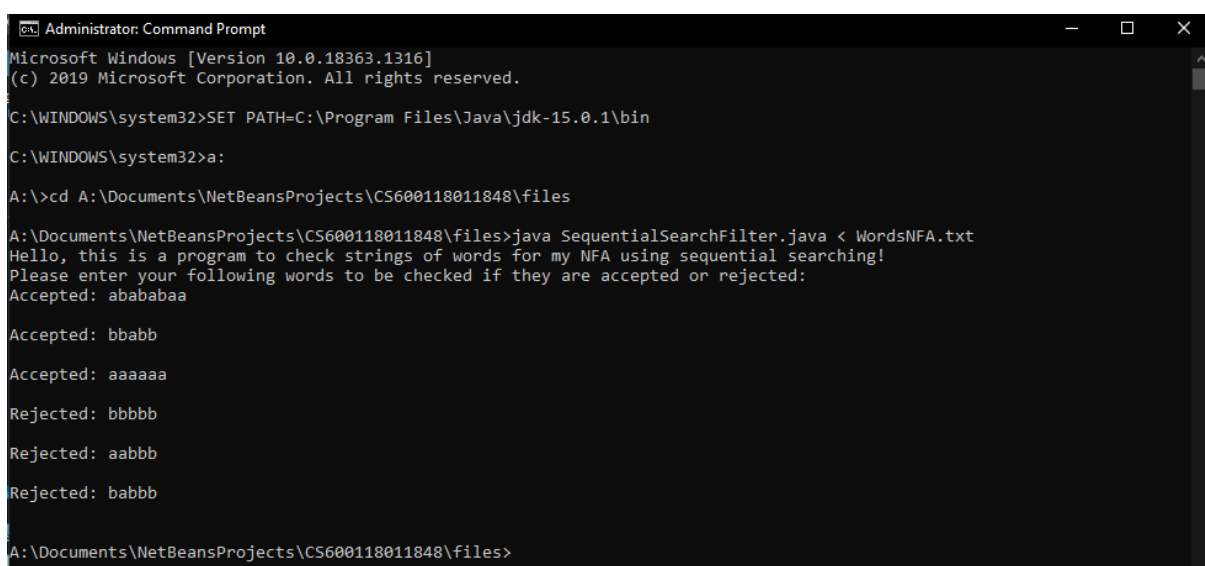


Figure 18: Using the WordsNFA.txt file to output results from the program.

Conclusion

To conclude, I found this individual implementation to be very challenging, however it has been a great learning experience. This is due to the implementation of the NFA, from modelling to coding within an IDE, to finally deploying and running through the command prompt.

Difficulties I had been starting to implement the code, to overcome this I referred to the workshop tasks which my tutor provided throughout previous weeks. Also, if I were not sure on certain aspects, I would communicate to my lecturer for my advice, this really benefitted me as I learnt a lot more about automata and therefore was able to understand how to implement them better. I also found working as a group really helped as we could support each other with both the group assignment and individual task if there was anything, we were unsure on.

Overall, this coursework was interesting and enjoyable to complete and I look forward to my next assignment within the module CS6001.

Source Code

Here is all the source code that was created during my time implementing the program. This is the source code taken from the NetBeans project, so therefore the packages are included.

SequentialSearchFilter Class

```
/*
 * Lauren Spruce
 * ID: 18011848
 */

import java.io.*;

/**
 * My Java application to show my SequentialSearchFilter
 * This application filters the input stream, whatever inputs are entered.
 * Any line of strings which are accepted or rejected will get outputted to the standard output.
 * Therefore the user can see the outputs for any word they input.
 */

public class SequentialSearchFilter {
    public static void main(String[] args)
        throws IOException {

        System.out.println("Hello, this is a program to check strings of words for my NFA using sequential
searching!");

        System.out.println("Please enter your following words to be checked if they are accepted or
rejected:");

        BufferedReader in = // standard input
```

```
new BufferedReader(new InputStreamReader(System.in)); //Allows the text to be read from an
input streamreader, which also allows files to be read.
```

```
//Allows buffering of characters so they can be easily read. Reads and echo's lines until the end of
the file
```

```
String s = in.readLine(); //Starts to read the strings as they are inputted
```

```
while (s!=null) {
```

```
    if (SequentialSearchNFA.accepts(s)) System.out.println("Accepted: " + s + "\n");// if bits accepted,
displays a 'accepted' message along with the string accepted otherwise skipped.
```

```
    else{
```

```
        System.out.println("Rejected: " + s + "\n");//If the string isn't accepted, will display reject
message along with the string.
```

```
    }
```

```
    s = in.readLine(); //Continues to read the strings as they are inputted
```

```
}
```

```
}
```

```
}
```

SequentialSearchNFA Class

```
/*
```

```
* Lauren Spruce
```

```
* ID: 18011848
```

```
*/
```

```
public class SequentialSearchNFA{
```

```
/*
```

```
* The transition function represented as an array.
```

```
* The entry at delta[s,c-'0'] is an array of 0 or
```

```
* more ints, one for each possible move from
```



```

* the state s on the character c.

*/

private static int[][][] delta = //3-dimensonal array
{ {{0,1},{0}}, {{2},{2,3}},{{1},{3}},{{},{}} };

// in this 3 dimensional we encode the states as a result of the transitions.

// Start state input a = [0][0][0] / [0][0][1] which represent the state we can go to {0,1} because if
we input a we can go state 0 or 1.

// Start state input b = [0][1][0] from start input b stays in 0.

/*

* Constants q0 through q4 represent states

*/

private static int q0 = 0; // initial state

private static final int q1 = 1; //The second state , state n1

private static final int q2 = 2; // The third state, state n2

private static final int q3 = 3; //the last state , state n3

private static final int q4 = 4; // the sink state is for when we are in the state 3 input a and b don't
go anywhere so the sink is for this cases

/**

* Test whether there is some path for the NFA to

* reach an accepting state from the given state,

* reading the given string at the given character

* position.

* @param s the current state

* @param in the input string

* @param pos index of the next char in the string

* @return true if the NFA accepts on some path

*/

private static boolean acceptsNext

(int state, String in, int pos) {

if (pos==in.length()) { // if no more to read

```

```

    return (state==q1 || state==q3); // in our NFA the final states are in q1 and q3
}

char c = in.charAt(pos++); // get char and advance the position
int[] nextStates;
try {
    nextStates = delta[state][c-'a']; // transition to the next position
    //c-a == difference of ASCII code c-a = 0 c-b = 1
}
catch (ArrayIndexOutOfBoundsException ex) {
    return false; // no transition, reject the entire string
}

// At this point after the transition, nextStates is an array
// of 0 or more next states. Try each move recursively;
// if it leads to an accepting state return true.

for (int i=0; i < nextStates.length; i++) {
    if (acceptsNext(nextStates[i], in, pos)) return true;
}

return false; // all moves fail, reject the entire string
}

/**
 * Test whether the NFA accepts the string.
 * @param in the String to test
 * @return true if the NFA accepts on some path
 */
public static boolean accepts(String in) {

```

```
    return acceptsNext(q0, in, 0); // start of the recursion in state q0 at char 0
  }

}

}
```