17/5/2022
Automata & Computability

Semester Project

Automata & Computability

Regular Expression to NFA Converter

Team #36

Reem Ahmed Sadek | 18P1490

Zead Hani Ali | 18P6573

I. Introduction

➢ Why we're doing this project?

Importance of Regular expression is that it's an important notation for specifying patterns. Each pattern matches a set of strings, so regular expressions serve as names for a set of strings. Programming language tokens can be described by regular languages. The specification of regular expressions is an example of a recursive definition.

Common applications include data validation, data scraping (especially web scraping), data wrangling, simple parsing, the production of syntax highlighting systems, and many other tasks.

➢ General description of REGEX, NFA & DFA

A. NFA Definition

NFA stands for non-deterministic finite automata. It is easy to construct an NFA than DFA for a given regular language.

The finite automata are called NFA when there exist many paths for specific input from the current state to the next state.

Every NFA is not DFA, but each NFA can be translated into DFA.
NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains ε transition.

B. Regular Expressions Definition

A regular expression (shortened as regex or regexp),
sometimes referred to as rational expression is a sequence of characters that specifies a search pattern in text.

Usually such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation. (Hopcroft)

C. DFA Definition

In the theory of computation, a branch of theoretical computer science,
a deterministic finite automaton (DFA)—also known as deterministic finite acceptor (DFA) is a finite-state machine that accepts or rejects a given string of symbols, by running through a state sequence uniquely determined by the string.

Deterministic refers to the uniqueness of the computation run. In search of the simplest models to capture finite-state machines.

II. Problem Definition

➢ Technical description of the problem

Finite State Automata are abstract models of computation. In other words, they are systems that take input in form of symbols which in our case considered to be the Regex and spit out a result or perform a computation.

To do this, they read symbols and move internally between states.
They can be described by 3 things:

- Starting state
- List of states
- Transitions between states

'Transitions between states' can also be thought of as a function that takes as arguments the current state and current symbol and returns what the next state should be.

The Problem we're addressing is a Regex that's entered to the program and the program shall be able to read the regex and convert it into a postfix regular expression then, convert it into an NFA with an option of converting it into a DFA.

At first, we formulated the problem to the main basic NFA's Structures which are Seen in our project as SingleStruct, Astric Struct, Concatenate Struct and Or Struct. Then the algorithm works by comparing each read-character to the input of the basic Nfa Structs and outputting the correspondent output.

➢ Problems we Faced

From the problems we faced during the implementation is the concatenation of different NFA's together to output the final graph of them all.

We solved it by using the concept of Epsilon – NFA's.
For Clarification, the following example AB

The NFA of this regex will be formulated as follows:

1) The Algorithm will create an NFA(1) for the Regex A
2) The Algorithm will create an NFA(2) for the Regex B
3) The Algorithm will concatenate both NFA's by adding and epsilon transition between the final state of NFA(1) and the Start state of the NFA(2).
4)

Also, we faced a problem with the import of Graphviz within the GUI and it was solved by adding a Clear Button to wipe out the shown NFA and proceed.

III. Approach

➢ Programming Language & Implemented Machines

The Programming Language used is Python.

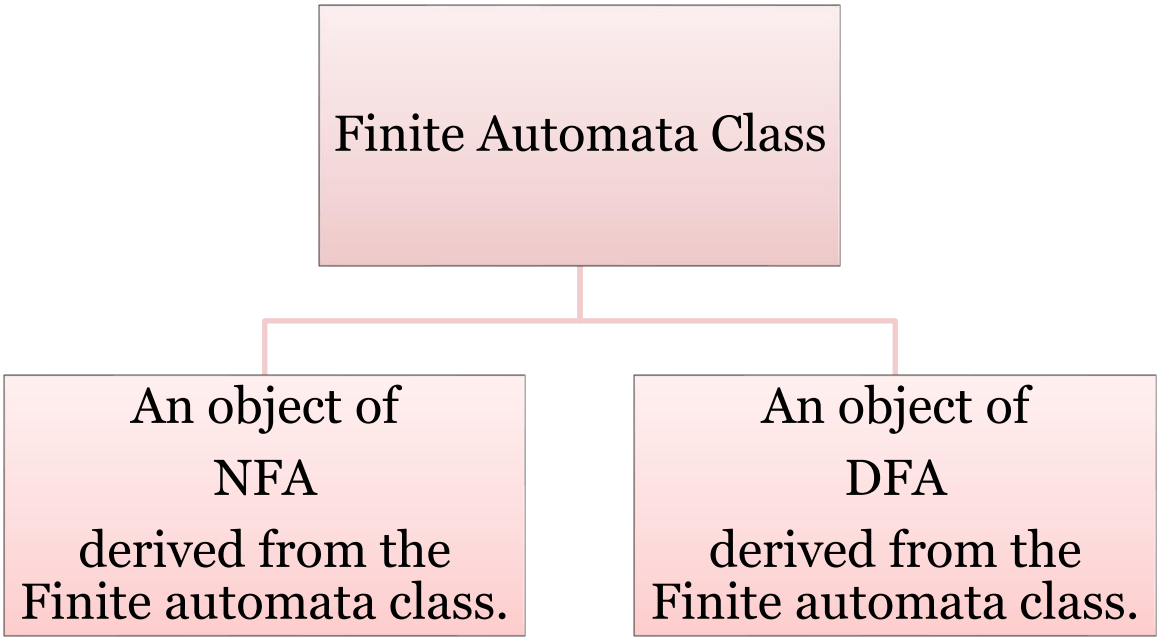
We used Object Oriented Programming (OOP) approach in this project, so everything is about classes and objects of such classes.

The program consists of two main classes which are the FA Class and the GUI Class, and inside the FA Class there exists other classes.

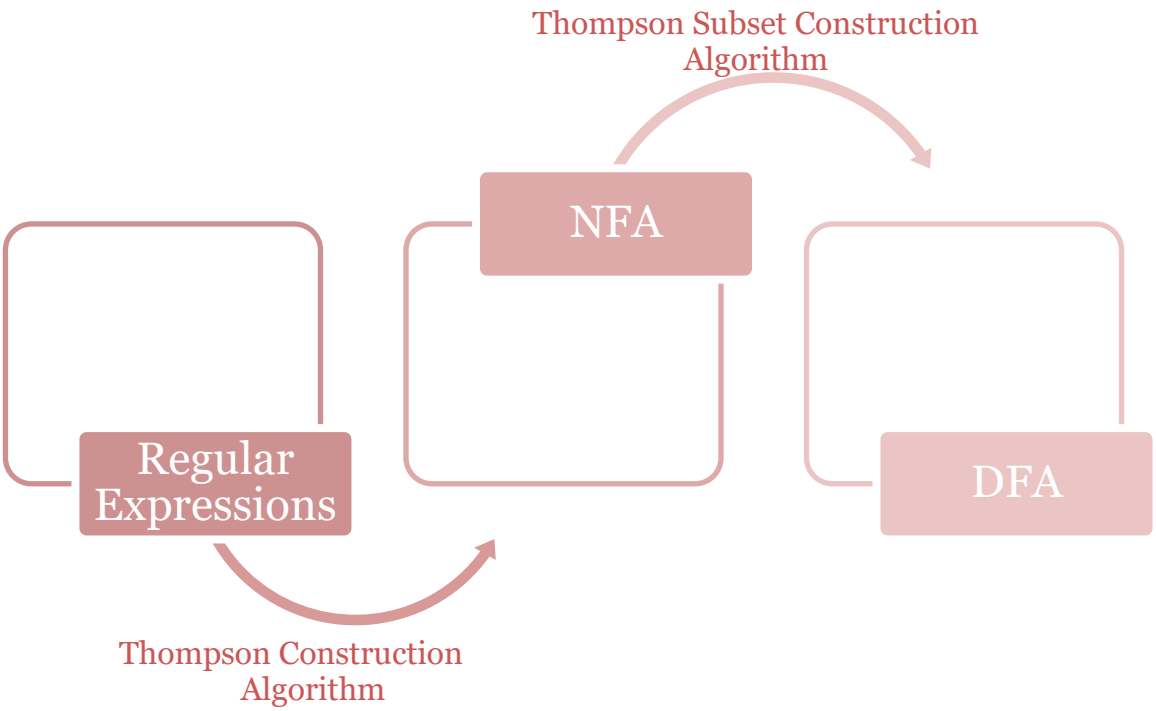
We used Tkinter in the development of the GUI Class and here's where the main function that start up the program exist as well.

We began by constructing a Finite Automaton class, from which objects of NFA and DFA can be derived. The general structure of the finite automaton class is as shown below.

The Drawing of the last graph of the Program is made using Graphviz and the output of graphviz can be spotted whether from the GUI window or the nfa/dfa.gv.png files.

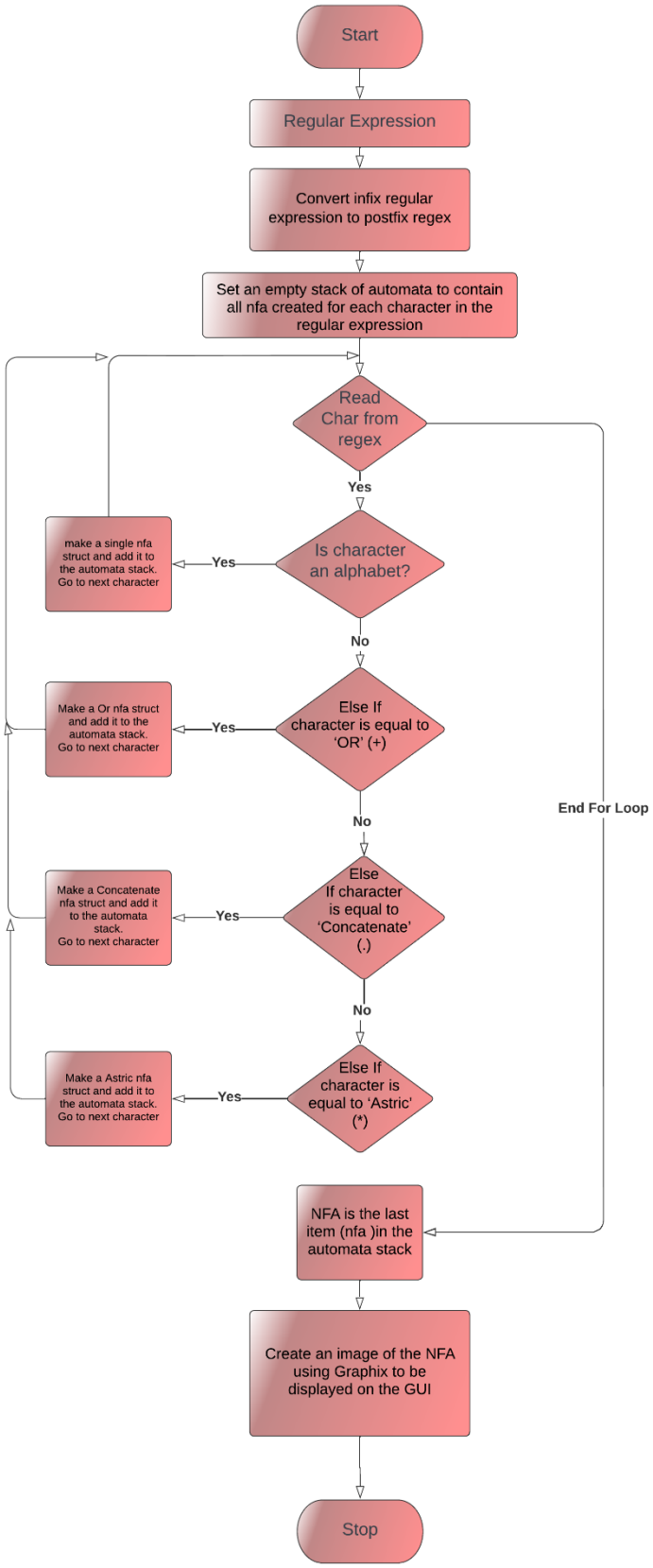


➢ Algorithms used in the Project

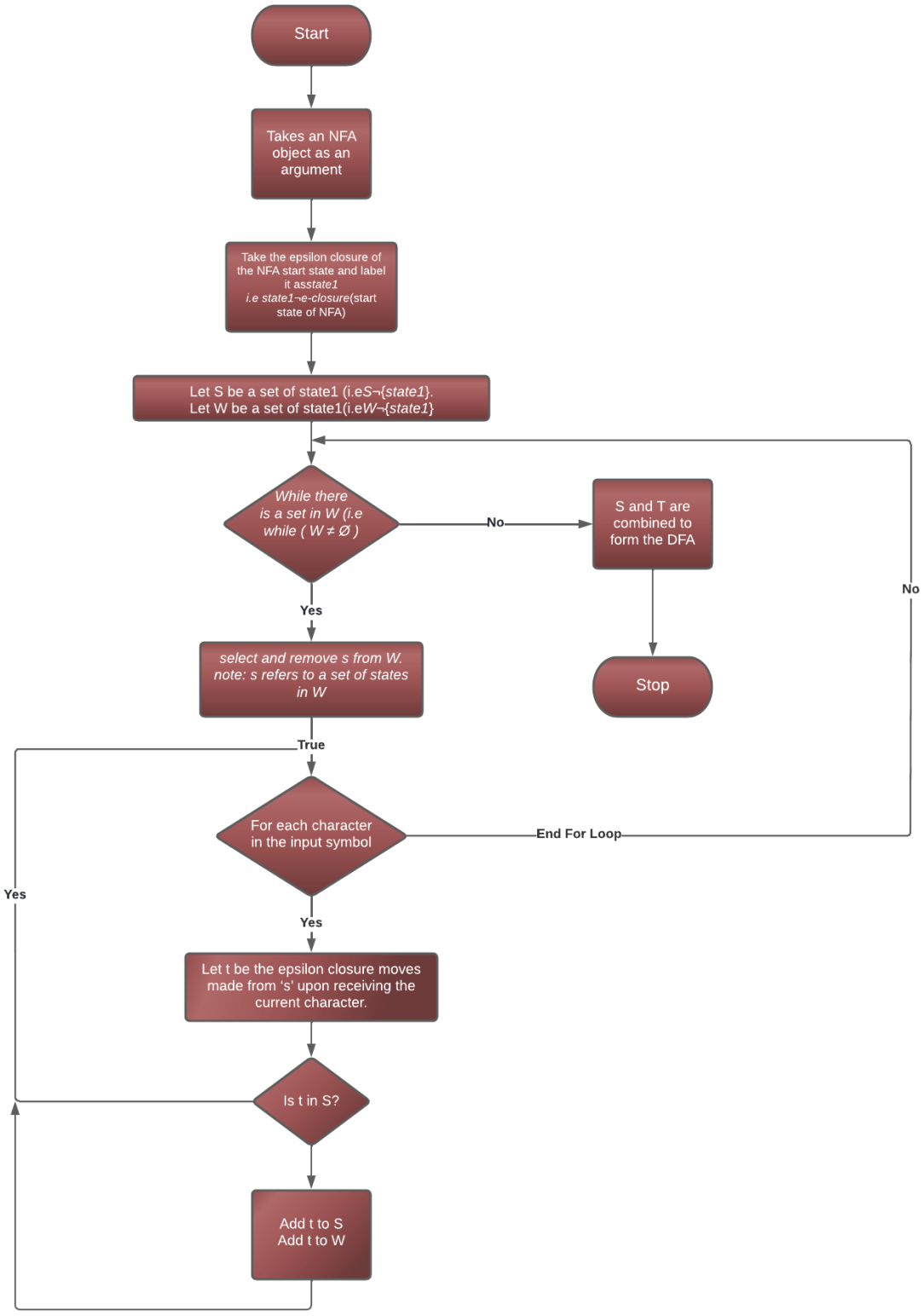


A. Thompson Construction Algorithm

As shown down there is the algorithm we used for the REGEX2NFA Converter



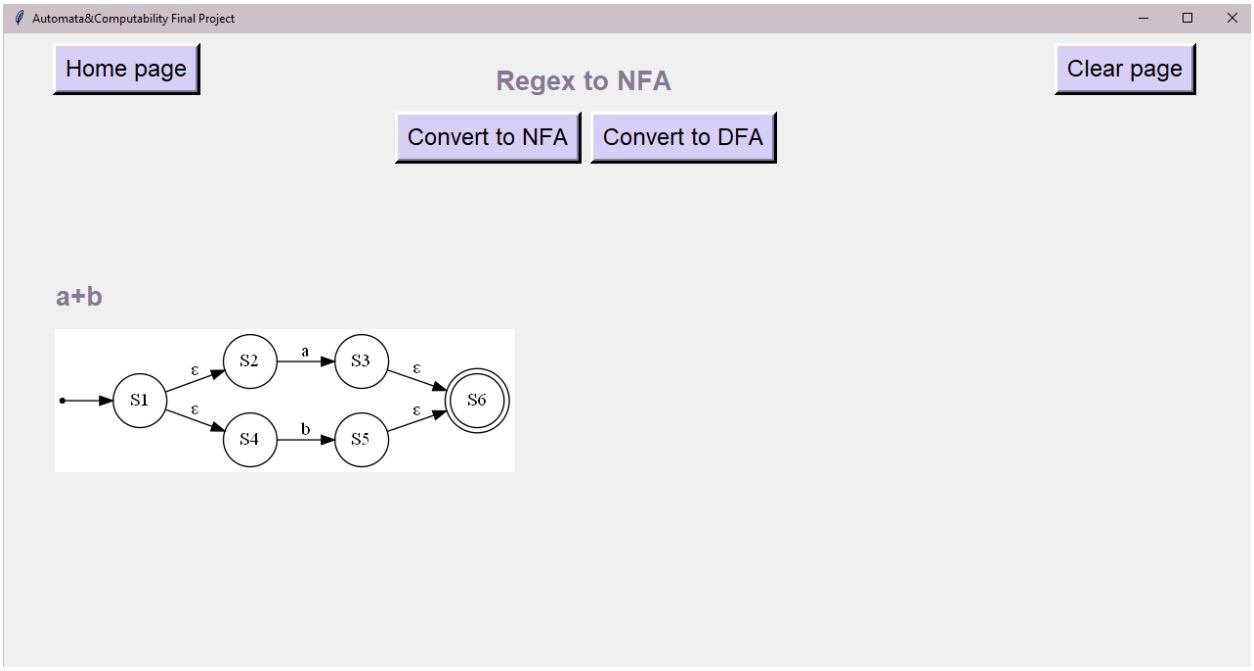
B. BONUS PART; Thompson subset construction algorithm



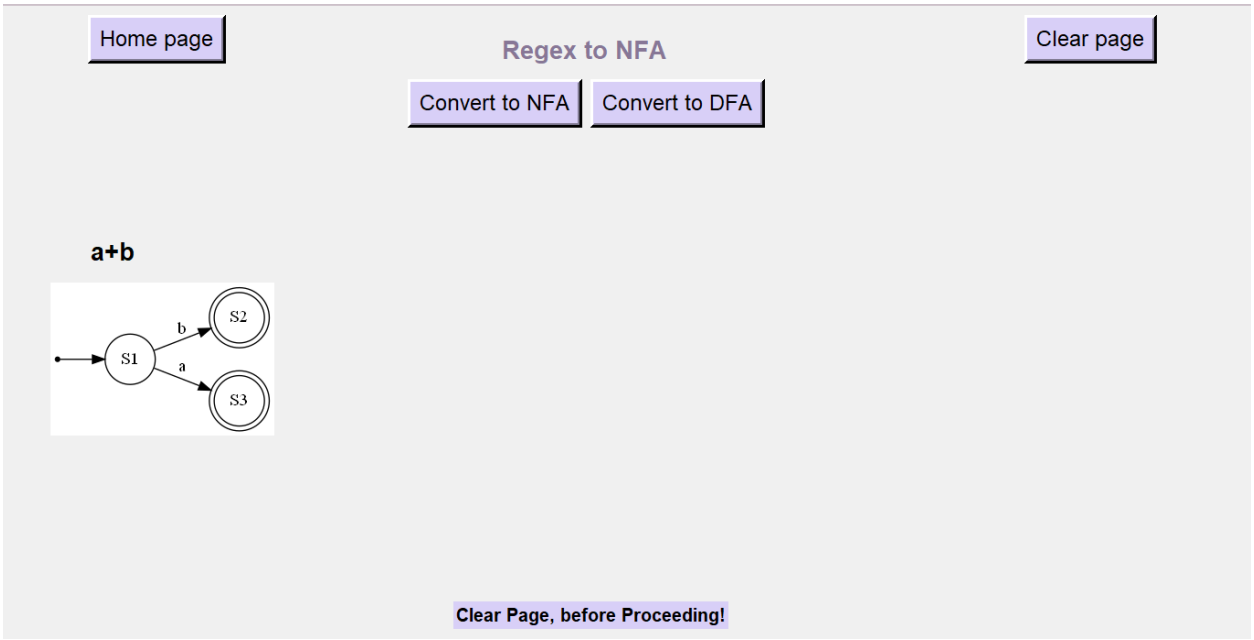
➢ Results and Analysis

We ran various examples on the project shown in the next section within the screenshots.

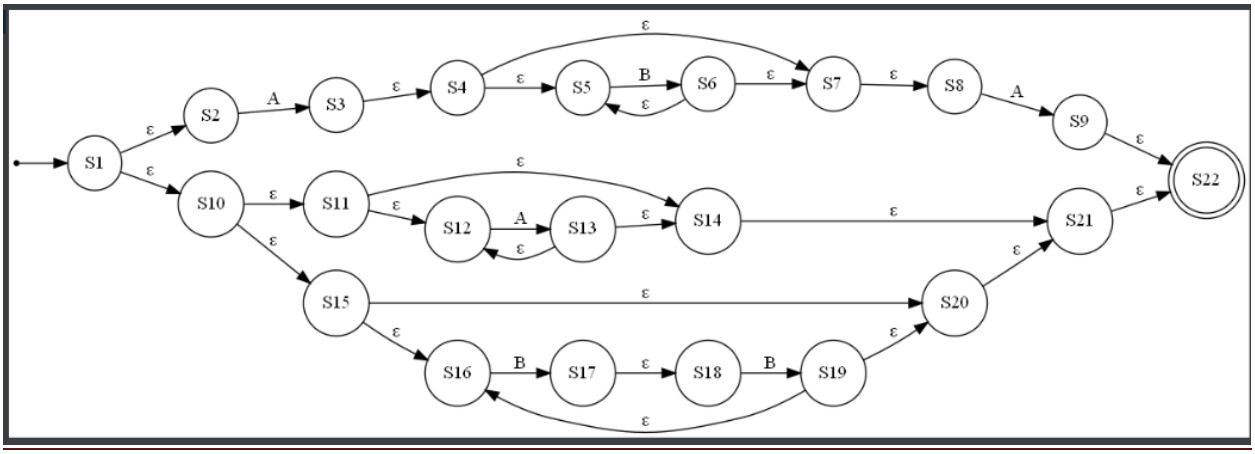
A+B: NFA



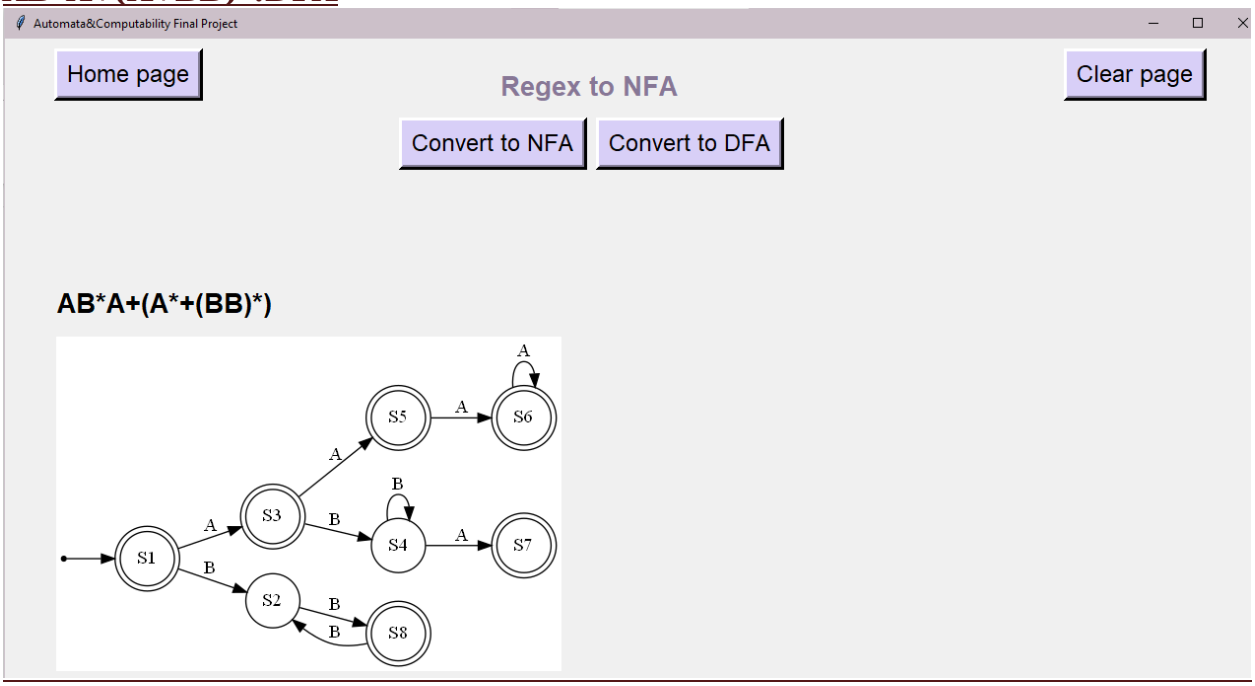
A+B: DFA



AB*A+(A+BB)*:NFA



AB*A+(A+BB)*:DFA



IV. Conclusion

We learnt from the project basically the importance of converting NFA to DFA and Regex to NFA.

Regular expressions are a way of expressing regular languages. In fact, the basic operation of a regex engine is to convert a regex pattern to either an NFA or DFA internally. Then, faced with a string, the N/DFA reads the symbols one by one and transitions between its states and finally spits out if the string of symbols matches the pattern/language or not.

The hint is in the expanded name. Computer programs generally need to know all possible transitions and states for a given state machine. A non-deterministic finite automaton can have a transition that goes to any number of states for a given input and state. This is a problem for a computer program because it needs precisely one transition for a given input from a given state. The process of converting NFA to DFA eliminates this ambiguity and allows a program to be made. NFA also suffers from combinatorial explosion. Reducing to a DFA can reduce state and transitions. (Hopcroft)

The purpose of an FSA is to check if a string of symbols can be accepted. If given a regular language that is described by an NFA and a DFA the DFA will take less time than the NFA to check if a string of symbols is accepted by the language. This is because at every state of a DFA, for a given input symbol, it's clear what next step to take. However, with an NFA, a depth-first search is required to check every possible combination. So we trade space for time when we use a DFA instead of an NFA!

V. References

- Chandra, A. (n.d.). Retrieved from Geeks for Geeks: https://www.geeksforgeeks.org/program-implement-nfa-epsilon-move-dfa-conversion/
- Channel, E. t. (2020, 8 4). Retrieved from https://youtu.be/HLOAwCCYVxE
- Hopcroft, J. U. (n.d.).
- Sintaha, M. (2015, Dec 22). Retrieved from https://youtu.be/oEraHUCwFVU
- vasugondaliya. (n.d.). Retrieved from Geeks For geeks: https://www.geeksforgeeks.org/converting-epsilon-nfa-to-dfa-using-python-and-graphviz/
- Bose, M. (n.d.). Retrieved from https://github.com/Megha-Bose/Automata-Theory-Conversions