**Automata Theory Semester Project**

**Converting Regular Expressions to Deterministic Finite Automata (DFA) or Non-Deterministic Finite Automata (NFA)**

**Finite Automaton Implementation**

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

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 shown below.

Finite Automata class

An object of DFA can be derived from the Finite automata class

An object of NFA can be derived from the Finite automata class

The algorithms used in this project are summarized below.

Regular expressions

Thompson Construction algorithm

NFA

Thomson subset construction algorithm

DFA

**Converting regular expression to NFA**

We used **Thompson construction algorithm** in our implementation of “regular expression to NFA”

Regular expression

Convert infix regular expression to postfix regex

Set an empty stack of automata to contain all nfa created for each character in the regular expression

For each character in regular expression

End of for loop

make a basic nfa struct and add it to the automata stack.

Go to next character

condition is true

Is character in alphabet?

yes

Make a line nfa struct and add it to the automata stack.

Go to next character

no

Else If character is equal to ‘line’ (|)

yes

Make a dot nfa struct and add it to the automata stack.

Go to next character

no

Else If character is equal to ‘dot’ (.)

yes

Make a star nfa struct and add it to the automata stack.

Go to next character

no

Else If character is equal to ‘star’ (\*)

yes

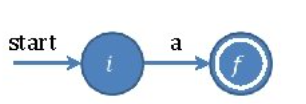
Create an image of the NFA using Digraph to be displayed on the GUI

NFA is the last item (nfa )in the automata stack

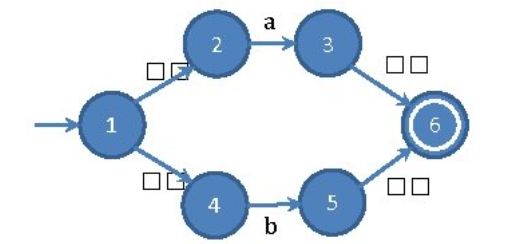
Note:

* “basic nfa struct” refers to converting a regular expression of the form ‘a’ to an NFA. That is

a -> NFA.

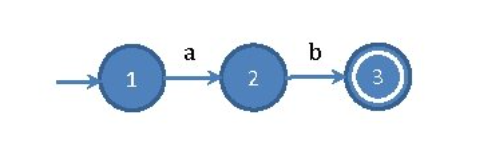


* “line nfa struct” refers to converting a regular expression of the form “a|b” to an NFA. (That is a|b -> NFA). This represents the union operation of regular expressions.



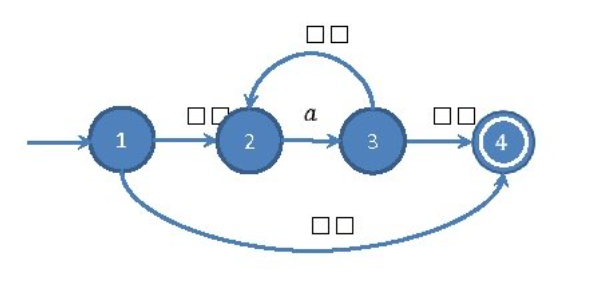
* “dot nfa struct” refers to converting a regular expression of the form “a.b” to an NFA. (That is

a.b -> NFA). This represents the concatenation operation of regular expressions.



* “star nfa struct” refers to converting a regular expression of the form “a.b” to an NFA. (That is

a\* -> NFA). This represents the kleene closure/operator.



Implementation of NFA to DFA using **Thompson’s subset construction algorithm**.

Takes an NFA object as an argument

Take the epsilon closure of the NFA start state and label it as *state1*

*i.e state1 ← ε-closure*(start state of NFA)

Let S be a set of state1 (i.e *S ←* { *state1*}.

Let W be a set of state1(i.e *W* *←* { *state1*}

*While there is a set in W (i.e while ( W ≠ Ø )*

End of while loop

S and T are combined to form the DFA

condition is true

*select and remove s from W.*

*note: s refers to a set of states in W*

For each character in the input symbol

(i.e *for each α ∈* Σ)

end of for loop

Let t be the epsilon closure moves made from ‘s’ upon receiving the current character.

(i.e *t ← ε-closure*(*Move*(*s,α*)).

*Let T[s,α] be t. (i.e T[s,α] ← t).*

*)*

*← t*

condition is true

Is t in S?

(i.e *if ( t ∈ S ))*

yes

no

Add t to S

Add t to W

Note:

* State1 in the above flowchart is a set of states
* S and W are sets of sets of states
* S and T form the DFA

The next flowchart shows an algorithm for checking if a string entered by a user matches with the regex defined.

For each character in string:

Set the start state of the dfa to be current state

replace epsilon in string by ‘@’ sign. This is because there is no epsilon sign on the keyboard

string

end of for loop

condition is true

Is the character epsilon?

skip

yes

Is the length of ‘state’ equal to zero?

Get all the states that can be traversed to, from the current state upon reading the character and save it to a variable (let’s call this variable ‘state’)

no

Return False

yes

no

Is current state in any of the final states of the dfa?

set current state to be the first element of ‘state’

Return True

yes

no

Return False

The operation of the graphical user interface (GUI) is shown in the flowchart below

User clicked on “convert to NFA” button?

User inputs regex

Main page

yes

no

Else if user clicked on “convert to DFA” button?

no

Stay on main page

yes

Go to DFA page

User clicks on ‘proceed to see DFA’ button?

User clicks on ‘proceed to see NFA’ button?

Go to NFA page

Display the DFA of the equivalent regex

yes yes

Clear page

Else if User clicks on ‘clear page’ button?

Display the NFA of the equivalent regex

no no

Else if User clicks on ‘clear page’ button?

yes yes

Stay on NFA page

Else if User clicks on ‘analyse regex’ button?

Else if User clicks on ‘analyze regex’ button?

Clear page

no

no

Take string from user

Go to ‘analysis’ page

Stay on DFA page

yes yes

Display “string doesn’t match with regex”

Display ‘string matches with regex’

String matches with regex?

Compare string with regex

User can check for another string or return to home page

User can check for another string or return to home page

yes no

The operation of the Graphic User Interface is as follows;

1. The user enters the regular expression
2. He then chooses to convert it to NFA or DFA
3. On both the NFA and the DFA pages, the user should click on “proceed to see NFA” or “Proceed to see DFA” button which will display the NFA or DFA of the regular expression he entered, depending on which one he chooses to convert to. He has the option to also clear the page (which clears all the information on the NFA or the DFA page) or click on “analyze regex” which takes him to the “Analysis page” where he can check for pattern matching. On the NFA and NFA pages, there are ‘home page’ buttons which will take him back to the home page of the app if he chooses to click on it.
4. On the analysis page, the user enters a string of text, and the program checks to see if the string the user entered matches with the pattern of the regular expression. If it matches, the program displays that the string the user entered matches with the pattern of the regex, in a green-background text. If it doesn’t match, the background color of the text is red and the user is told the string he entered doesn’t match with the regex. On this page, the user can check for the matching of strings for an unlimited number of times and he can finally return back to the home page by clicking on the home page button.