Colin Dutra
CS454
12/6/2018

Regex Parsing & NFA Construction

String Membership in Regular Expressions through the construction of Epsilon free NFAs

Description of Program:
This program takes two things, a string (w) and a regular expression(L). Our program takes our regular expressions, and using the procedures we learned in class, converts the regular expression into an equivalent NFA. Then, we turn that NFA into an epsilon free NFA.

Next, we take the sting W, and feed it into our NFA, and test it's acceptance.

The entire program is fed into a GUI which makes it easy to use.

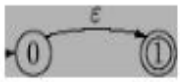
For the sake of demonstration, and examples of the expected language format, the GUI fields are populated with accepting cases.

Core Algorithm:
The majority of this project is based on *Thompson's Algorithm*, which is the algorithm used to convert regular expressions into equivalent NFAs.

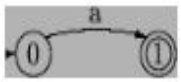
Converting a regular expression to a NFA - Thompson's Algorithm

We will use the rules which defined a regular expression as a basis for the construction:

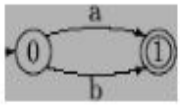
1. The NFA representing the empty string is:



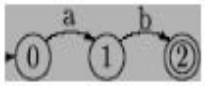
2. If the regular expression is just a character, eg. a, then the corresponding NFA is :



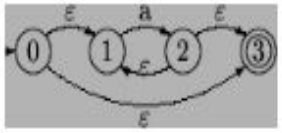
3. The union operator is represented by a choice of transitions from a node; thus a|b can be represented as:



4. Concatenation simply involves connecting one NFA to the other; eg. ab is:



5. The Kleene closure must allow for taking zero or more instances of the letter from the input; thus a* looks like:



Source: http://www.cs.may.ie/staff/jpower/Courses/Previous/parsing/node5.html

Data Structures & Libraries Used:

Libraries:

- Random: For “demo” values in GUI fields.
- Sys: For exiting the program when we need to.
- Tkinter: Used for creating and editing the GUI.

Python Features:

Dictionaries: Used for Transition Tables

Set: For holding states.

Data Structures:

Stack: Used for the processing of the regular language.

Basic Description of Our Solution:

On startup, our program opens up our UI window, with demo values already loaded in. However, the user is free to enter in new values to test. Upon hitting the “check” button our algorithm begins.

First, we take two variables from the UI, the regex and the string. From there, we trigger our NFAfromRegex function, which takes the input regular expression as its parameter From here, the NFA is built using our stack and operator processing, and our individual operator functions (star, dot, bar, etc).

After this, in our check function we call “eFreefromNfa” which does what you expect, the e-closure of the NFA.

Then, we call the string acceptance function on our newly formed epsilon free NFA, and pass the result to our UI to display whether or not it is accepted.

Core Functions and Classes:
Static Methods for Building from Regex Symbols:
There is a static method for the OR bar, the AND dot and the Kleene Star..
Stack Functions:

addOpToStack and processOP process the regex fed to it, and determine any errors in entry before. processOp triggers the starAutomata, dotAutomata, and barAutomata functions in the BuildAutomata class.

buildNFA:
When we create an NFA object, this is called at the end of our initialization to actually create it.

Check():

Check is for all intents and purposes our “main” function. It pulls from our two entry fields, makes an NFA, makes an epsilon free nfa from that, then tests the strings acceptance in that NFA,and displays the result in our GUI.

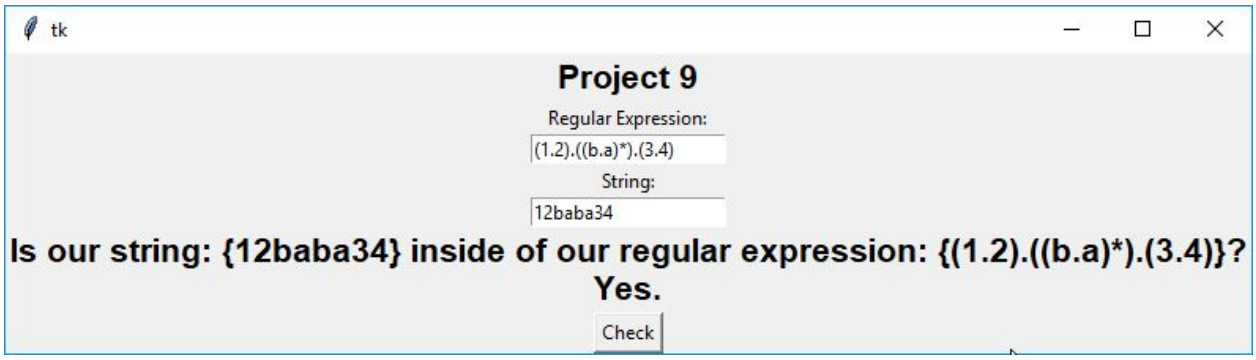
Navigating the UI:

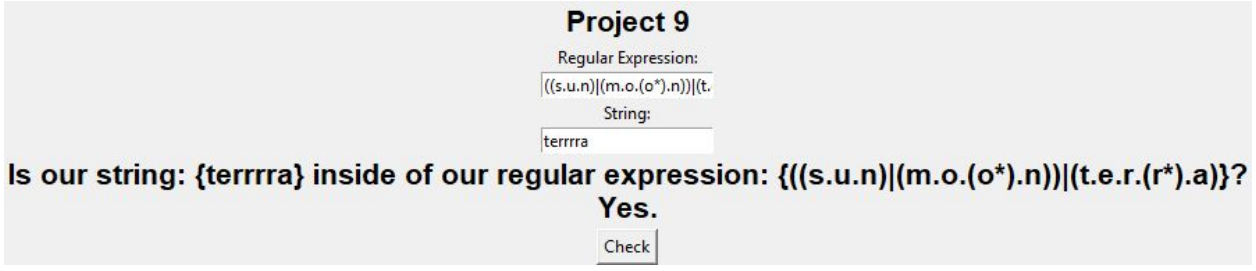
The first field in the UI is for the regular expression, be aware that for the concatenation operator, you have to do A.B not AB.

The second field is for the string you'll be running through the regular expression.

Once both are filled out, hit the "Check" button to see if your regular expression accepts your string.

Sample Input & Output Screenshots:





Summary and Conclusions:
The concept that I grasped the most during this semester was converting regular expressions into Automata, I found it really satisfying watching it grow and unfurl.

With this project, I was very satisfied with the end product, especially learning how to implement a GUI that works through multiple iterations of the programming.

Some extensions of the program would include:
Doing the inverse, taking an NFA and turning into a regular expression using image processing or a similar apparatus. If I had more time, I would also implement much better error handling, as right now it quits out (using exceptions) if it encounters any issues.

Another possible extension which would be very cool, would be to show a timelapse image of the algorithm at work, showing the NFA being built in front of the user's eyes.

All in all, this was a tough but rewarding project, and I feel that I learned a lot more about python as a language through it.