Mission:	Project Part 1 (NFA implementation)
DID:	Software Test Document (STD)
PREPARED by:	s99 Team
	(
	Samuel Sessums
	)
DATE:	10/19/2021

Table of Contents

1. Scope

1.1 Identification

The purpose of this document is to explain how this software is implemented and a detailed report of testing performed by members of the team.

1.2 System Overview

Part I of this project requires an implementation to simulate Deterministic Finite Automata (DFA) with input files and user supplied words . However, extra credit has been assigned to also simulate Non-Deterministic Finite Automata (NFA). This program is meant to be used as a console application.

1.3 Definitions, Acronyms and Abbreviations

DFA - Deterministic Finite Automata(automaton)

NFA - Nondeterministic Finite Automata(automaton)

1.4 References

aboSamoor, & joeld. (1957, February 1). *How to print colored text to the terminal*. Stack Overflow. Retrieved November 3, 2021, from https://stackoverflow.com/questions/287871/how-to-print-colored-text-to-the-terminal.

cyberzhg. (n.d.). Regex => NFA => DFA. Retrieved November 3, 2021, from https://cyberzhg.github.io/toolbox/nfa2dfa.

Goyvaerts, J. (n.d.). How to find or validate an email address. Retrieved November 3, 2021, from https://www.regular-expressions.info/email.html.

iCrus, iCrusiCrus 1, & user513418user513418. (1962, April 1). *FSM for email address format validation*. Stack Overflow. Retrieved November 3, 2021, from https://stackoverflow.com/questions/21134597/fsm-for-email-address-format-validation.

Make a readme. Make a README. (n.d.). Retrieved November 4, 2021, from https://www.makeareadme.com/.

patorjk. (n.d.). Text to ASCII art generator (TAAG). Retrieved November 4, 2021, from http://patorjk.com/software/taag/#p=display&f=Graffiti&t=Type%20Something%20.

Rodger, Susan JFLAP. (2018). Retrieved from https://www.jflap.org/jflaptmp/

Signal - set handlers for asynchronous events¶. signal - Set handlers for asynchronous events - Python 3.10.0 documentation. (n.d.). Retrieved November 4, 2021, from

2. Program Implementation

This is a brief high-level explanation of the program's NFA implementation i.e. not all specifics are given to how the program functions. After the file has been parsed and verified to be correct, a NFA is constructed using the standard five tuple NFA arguments and an additional type description argument. Once constructed, the automaton can parse words. The NFA simulation is recursive. If there are no more characters of the word to parse, no more epsilon transitions, and the last state it was in is an accept state, then the word is a part of the automatons language; otherwise, it is not part of language and is rejected. However, if there are more characters of the word left, then check if the current state has the first letter of the word in it's list of transitions to take. If the state doesn't have that character as a transition or no epsilon transitions, then the word is rejected; otherwise, the current state changes to the new state and looks at the next character to parse. Side note: If the character '~' is in the NFA's alphabet then it will be treated as a space ' '. This can be changed from the global variable SPACE.

3. Unit Testing

For DFA testing, sections will be divided by a file and a table describing the word being parsed, the path taken, and whether the word is accepted or rejected by the respective automaton.

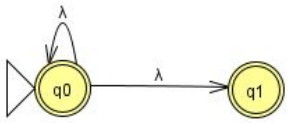
3.1 NFA Testing

Unfortunately, tests are non-exhaustive and won't prove program correctness; however I will present several tests where NFAs range from any combination of small or large to simple or complex. I explored several methods to create NFAs, but ultimately chose to use Susan Rodger's excellent program JFLAP and cyberzhg's web tool to create finite state machines. This allowed me to test more examples that should be correct than what is shown below.

3.1.1 Standard Testing

3.1.1.1 File 1 - “nfa1.txt”

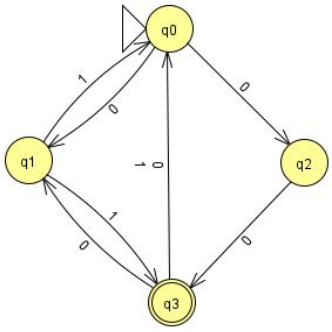
REGEX = ε



Word(String)	Path taken	Accept or Reject
Empty string (“”)	Q0	Accept
Ithoughtthiswouldwork	Q0 Q1	Reject
apparently	Q0 Q1	Reject
not.A++	Q0 Q1	Reject

3.1.1.2 File 2 - “nfa2.txt”

REGEX = ((01)*(01|00)(01)*(0|1|01))*(01)*(01|00)(01)*

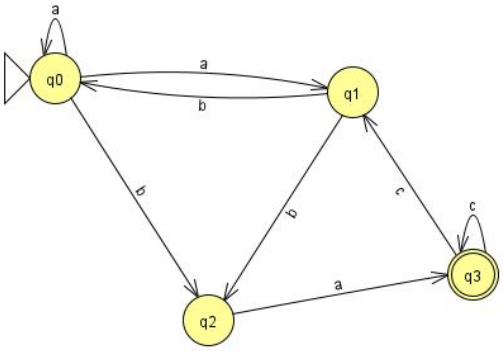


Word(String)	Path taken	Accept or Reject
Empty string (“”)	Q0	Reject
0	Q0 Q2	Reject
00	Q0 Q2 Q3	Accept
01	Q0 Q1 Q3	Accept
0101111101	Q0 Q2	Reject
01010101	Q0 Q1 Q0 Q1 Q0 Q1 Q0 Q1 Q3	Accept
0000110	Q0 Q2 Q3 Q1	Reject
010101010321	Q0 Q2	Reject

00001000000021	Q0 Q2 Q3 Q1	Reject
jkadsh	Q0	Reject
0111010101246541321	Q0 Q2	Reject
0101010101010101010101	Q0 Q1 Q0 Q1 Q0 Q1 Q0 Q1 Q0 Q1 Q0 Q1 Q0 Q1 Q0 Q1 Q0 Q1 Q0 Q1 Q0 Q1 Q3	Accept
010010100010001001	Q0 Q1 Q0 Q2 Q3 Q0 Q1 Q0 Q2 Q3 Q1 Q0 Q2 Q3 Q1 Q3 Q0 Q1 Q3	Accept

3.1.1.3 File 3 - “nfa3.txt”

REGEX = ((a|ab)*(b|ab)a(c|cba)*cb)*(a|ab)*(b|ab)a(c|cba)*



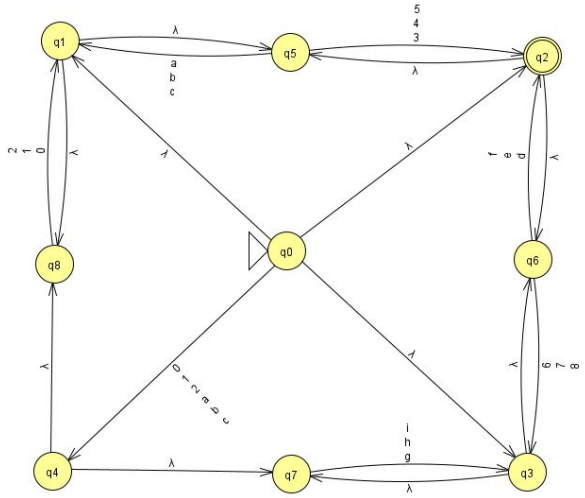
Word(String)	Path taken	Accept or Reject
Empty string (“”)	Q0	Reject
aaaaaaacb	Q0 Q1	Reject
aba	Q0 Q0 Q2 Q3	Accept
abacccccbbba	Q0 Q1 Q2 Q3 Q3 Q3 Q3 Q3 Q3 Q3	Reject
ababababa	Q0 Q1 Q0 Q1 Q0 Q1 Q0 Q0 Q2 Q3	Accept
ccbc	Q0	Rejected

cbabbaa	Q0	Rejected
abbabbaabab	Q0 Q1 Q2	Rejected
ababccchv	Q0 Q1 Q2 Q3	Rejected
aaaaaaaaaabbbbccccccc	Q0 Q1	Rejected
acccccaabaa	Q0 Q1	Rejected
2983479283	Q0	Rejected
iuskdfjh	Q0	Rejected

3.1.1.4 File 4 - “nfa4.txt”

REGEX =

(ε|(a|b|c)*(3|4|5)|(6|7|8)*(d|e|f)|(ε|0|1|2|a|b|c|(6|7|8)*(6|7|8))(g|h|i|(g|h|i)(6|7|8)*(6|7|8))*(g|h|i)(6|7|8)*(d|e|f)|(ε|0|1|2|a|b|c|(a|b|c)*(a|b|c))(0|1|2|(0|1|2)(a|b|c)*(a|b|c))*(0|1|2)(a|b|c)*(3|4|5))((a|b|c)*(3|4|5)|(6|7|8)*(d|e|f)|(6|7|8)*(6|7|8(g|h|i|(g|h|i)(6|7|8)*(6|7|8))*(g|h|i)(6|7|8)*(d|e|f)|(a|b|c)*(a|b|c)(0|1|2|(0|1|2)(a|b|c)*(a|b|c))*(0|1|2)(a|b|c)*(3|4|5))*

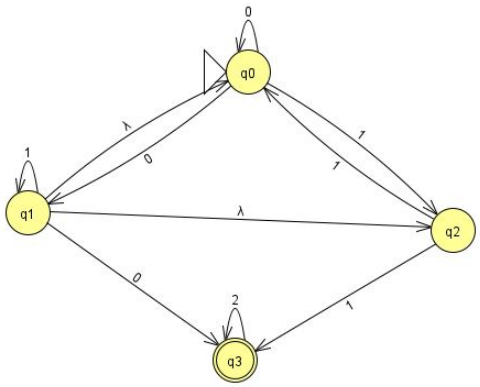


Word(String)	Path taken	Accept or Reject
Empty string (“”)	Q0 Q2	Accept
abcdef233	Q0 Q4 Q8	Reject
834792837	Q0 Q3 Q7	Reject

abd1bc	Q0 Q4 Q8	Reject
abc38	Q0 Q4 Q8	Reject
ab	Q0 Q4 Q8	Reject
3d3d3d	Q0 Q3 Q7	Reject
3e	Q0 Q3 Q7	Reject
3847	Q0 Q3 Q7	Reject

3.1.1.5 File 5 - “nfa5.txt”

REGEX = (0|01*|(1|01*)1)*(01*0|(1|01*)1)2*

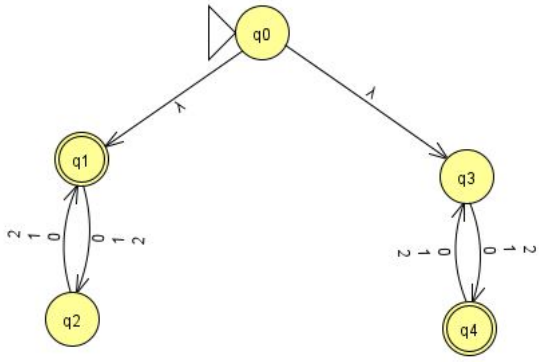


Word(String)	Path taken	Accept or Reject
Empty string (“”)	Q0	Reject
11	Q0 Q2 Q3	Accept
00	Q0 Q1 Q3	Accept
000000022222	Q0 Q0 Q0 Q0 Q0 Q0 Q1 Q3 Q3 Q3 Q3 Q3 Q3	Accept
01001012	Q0 Q1 Q1 Q3	Reject
010100103	Q0 Q1 Q1 Q3	Reject
1111111	Q0 Q2 Q3	Reject
1122222	Q0 Q2 Q3 Q3 Q3 Q3 Q3	Accept

	Q3	
00022222111	Q0 Q1 Q3	Reject

3.1.1.6 File 6 - “nfa6.txt”

REGEX = ε|(0|1|2)((0|1|2)(0|1|2))*(0|1|2)|(0|1|2)((0|1|2)(0|1|2))*

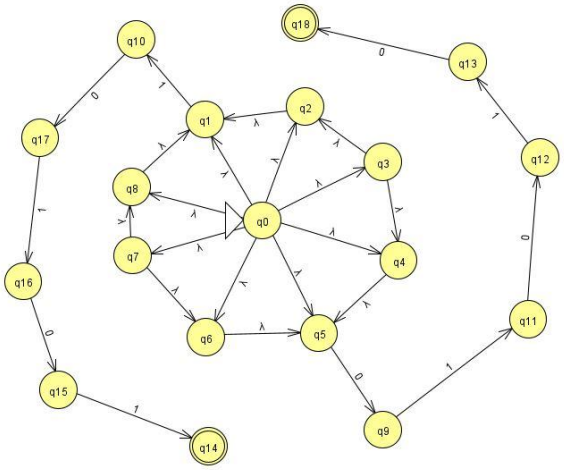


Word(String)	Path taken	Accept or Reject
Empty string (“”)	Q0	Accept
000111010101010	Q0 Q3 Q4 Q3 Q4 Q3 Q4 Q3 Q4 Q3 Q4 Q3 Q4 Q3 Q4 Q3 Q4	Accept
1000000000001110	Q0 Q1 Q2 Q1 Q2 Q1 Q2 Q1 Q2 Q1 Q2 Q1 Q2 Q1 Q2 Q1 Q2 Q1	Accept
111111111111111	Q0 Q3 Q4 Q3 Q4 Q3 Q4 Q3 Q4 Q3 Q4 Q3 Q4 Q3 Q4 Q3 Q4	Accept

1222	Q0 Q1 Q2 Q1 Q2 Q1	Accept
0	Q0 Q1	Accept
00	Q0 Q1 Q2 Q1	Accept
11111	Q0 Q3 Q4 Q3 Q4 Q3 Q4	Accept
sdlkfjsldkfj	Q0 Q3	Reject

3.1.1.7 File 7 - “nfa7.txt”

REGEX = (1|1|1|1)0101|(0|0|0)1010



Word(String)	Path taken	Accept or Reject
Empty string (“”)	Q0	Reject
1111	Q0 Q8 Q1 Q10	Reject
00010	Q0 Q8 Q1	Reject
dasdfd	Q0 Q8 Q1	Reject
12312	Q0 Q8 Q1 Q10	Reject
10101	Q0 Q1 Q10 Q17 Q16 Q15 Q14	Accept
01010	Q0 Q3 Q4 Q5 Q9 Q11 Q12	Accept

	Q13 Q18	
00000	Q0 Q8 Q1	Reject
11111111111	Q0 Q8 Q1 Q10	Reject