Quiz 1

EECS665 - Compiler Construction
2021, Fall

Name: _____ Student ID: _____

Do not open until Instructed!

Before the quiz starts:

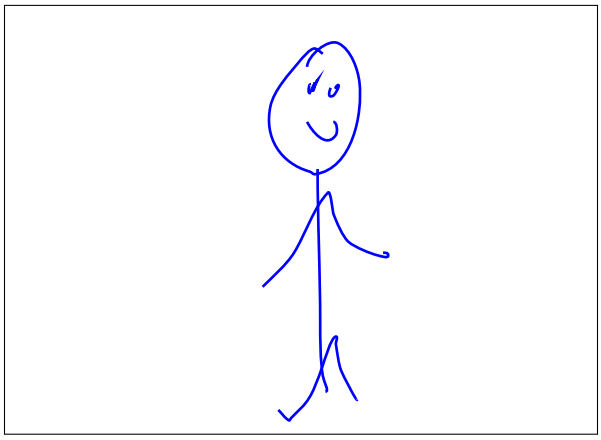
- Read all instructions on this page
- Write your name and student ID on this page
- Retrieve your page of notes (if you have it) and writing materials
- Put all other materials away and silence your devices

After the quiz starts:

- Write student ID (**not** your name) on **all** subsequent pages
- If you feel a question is wrong or impossible, notify course staff
- Announcements / corrections will appear on the projector, remember to check
- You may turn in your quiz early and leave when done
- Work quickly, and move on if you are stuck

Total Questions: 5
Total Pages: 6
Total Points: 50

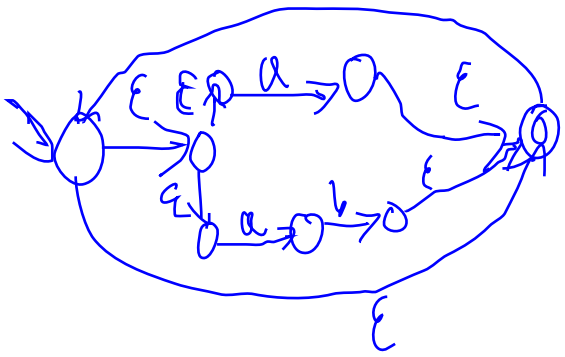
Feel free to put a drawing of yourself in the box below



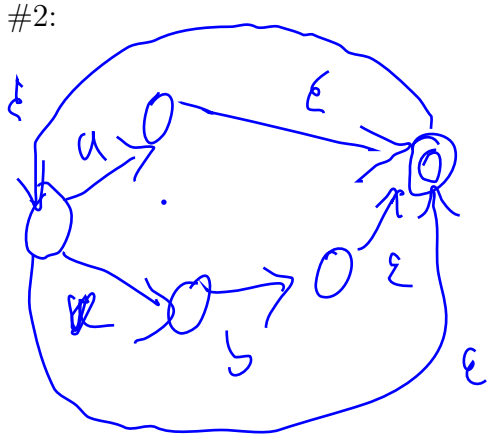
Question 1 (10 Points)

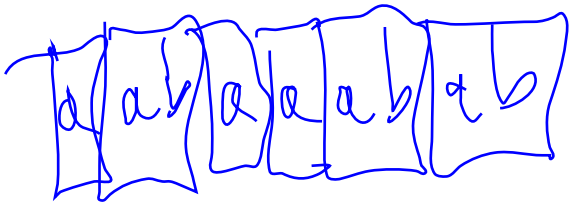
Give **two** finite automata (DFAs and/or NFAs) that both recognize the regular expression (a|ab)*

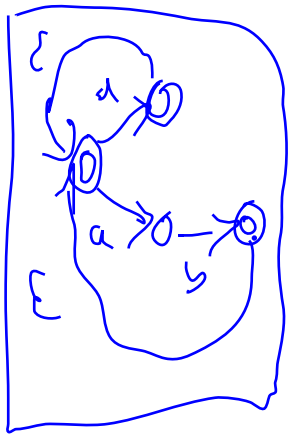
Machine #1:



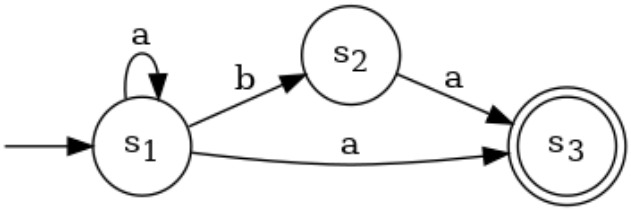
Machine #2:



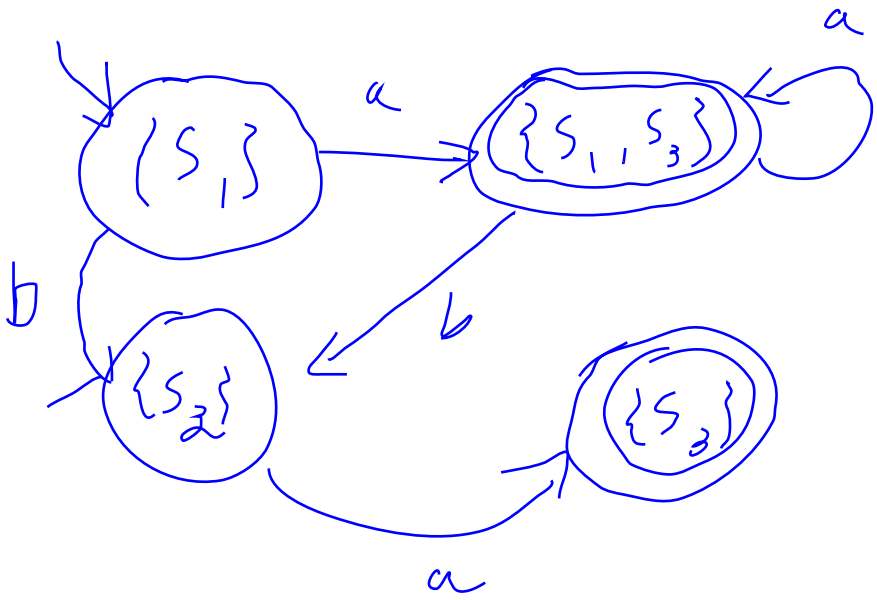




Question 2 (10 Points)



Show the equivalent DFA for the above NFA (you may use the Rabin-Scott Powerset Construction, or just eyeball it).



Question 3 (10 Points)

Consider a Flex specification with the following rules section

```
a[c-f]c  { std::cout << "1";}
a[.]c    { std::cout << "2";}
a[cf]c+  { std::cout << "3";}
a.c      { std::cout << "4";}
[ ]      { std::cout << "5";}
.        { std::cout << "6";}
```

Write the output of the generated lexer on the string

`acadccafcc`

6616*

3

Question 4 (10 Points)

Let WaCkYmaTh be the language of mathematical expressions allowing only the operators **plus**, **minus**, and **times** and the operand **intlit**.

Part I

Create a context-free grammar for WaCkYmaTh expressions. The grammar should be unambiguous and (unlike "traditional" math) enforces the following structure:

- times is the **lowest** precedence operator
- plus is the **highest** precedence operator
- all operators are right associative

E ::= T times E { $$ = $1 * $3 }
| T { $$ = $1 }
T ::= F minus T { $$ = $1 - $3 }
| F { $$ = $1 }
F ::= Q plus F { $$ = $1 + $3 }
| Q { $$ = $1 }
Q ::= intlit { $$ = $1 } LHS.trans = intlit.value

LHS.trans = T.trans * E.trans

Part II

Add a syntax-directed definition for WaCkYmaTh that translates an expression to its mathematical value. Assume that the intlit token has a value field that returns it's numeric value. Give an expression and show that it's value is different than that of "traditional" math.

regular
2 * 2 + 3
~~8~~
~~10~~
7

wacky
2 * 2 + 3
10

Question 5 (10 Points)

The tokenizers we create in class process input until all machines fail, then rewind the stream to a point where at least 1 machine was in an accepting state. In the examples given in class, the machine only ever has to rewind by 1 character.

Part I

Is it possible to define a set of token patterns where the tokenizer has to rewind by **more than 1** token?

yes

Part II

If your answer to Part I was yes, show a set of token patterns where it might need to rewind by more than 1 character and show an input where that rewind would happen. If your answer to Part I was no, describe why the machine never needs to rewind by more than 1 character.

token patterns

a+b

a

input

aaaadada