Calculate FIRST sets for the following Grammar

S ::= lpar X rpar
X ::= id comma X
| ε

Q1

Question 1 (10 Points)

Assume, for this question, that you are the captain of a pirate ship.

Part I

Your first mate claims that **there exists** an NFA with 3 states, and that **there exists** an equivalent DFA with only 1 state. Is your first mate correct? If so, give an example of such an NFA / DFA pair. If not, explain why no such pair could exist.

Part II

Your helmsman claims that **there exists** an NFA with 3 states, and that **any** equivalent DFA **must have** at least 8 states. Is your helmsman correct? If so, show the NFA and it's equivalent DFA. If not, explain why no such NFA could exist.

A B C

{A, B, C}

{A, C} {A, B}, {B, C}

{A} {B} {C}

{}

Q

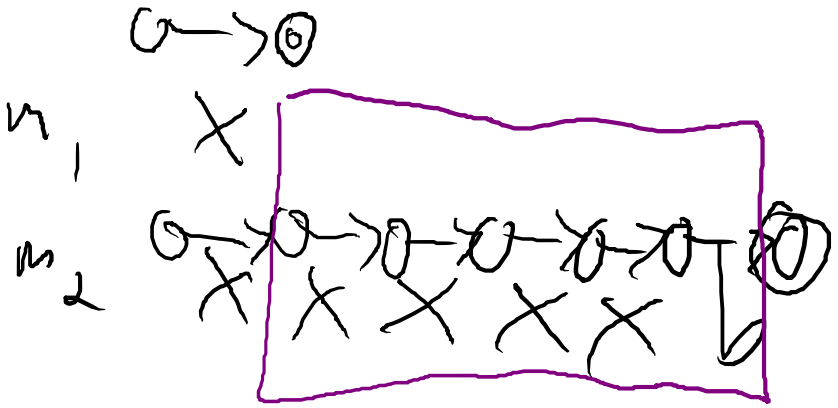
$2^{|Q|} - 1$

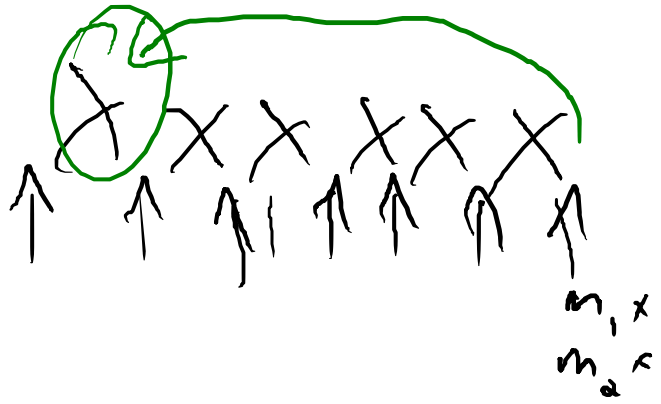
Q2

Question 2 (10 Points)

We described a method by which DFAs could be used to create a tokenizer (i.e. a translator from a stream to characters to a stream of tokens). The tokenizer had to backtrack over the input string even after an accepting state had been been found for one of the token DFAs.

Write a series of token languages and an input string that would cause the tokenizer to backtrace by at least 2 characters. Explain why this backtracking happens.

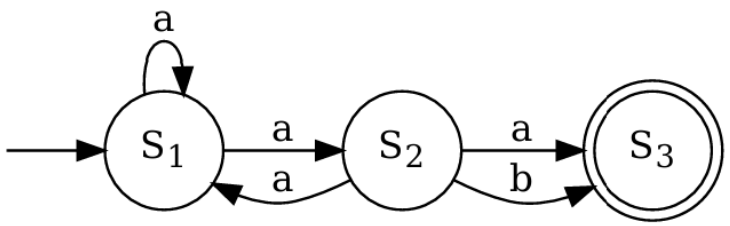


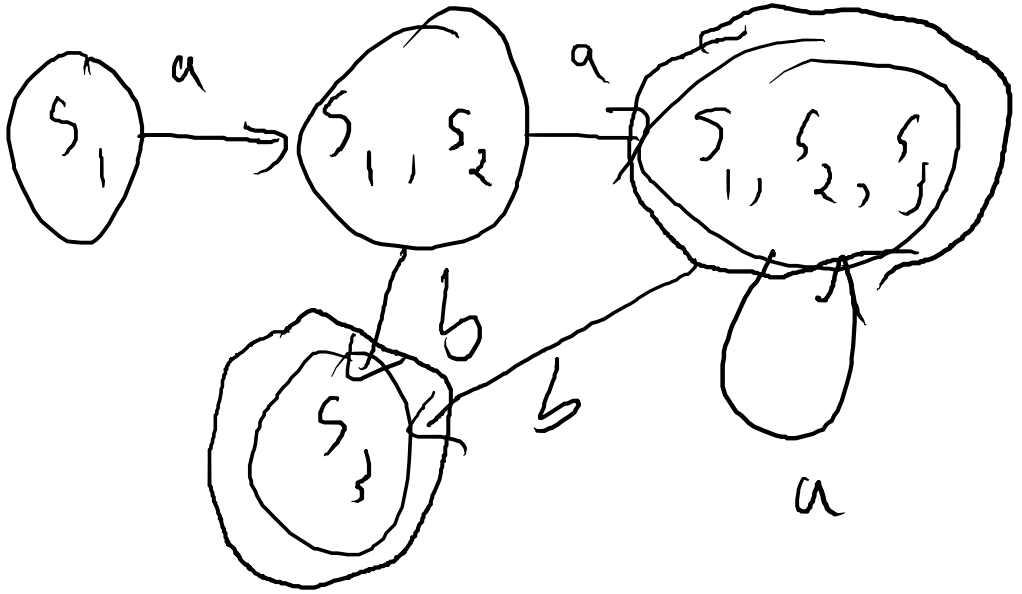


Q3

Question 3 (10 Points)

Create an equivalent DFA to the given NFA (e.g., use the Rabin-Scott powerset construction):





Q4

Question 4 (10 Points)

The language ***of*** all regular expressions can itself be expressed as a context free grammar! Imagine that a tokenizer has already built that uses the tokens

star, **or**, **lpar**, **rpar**, and **char** for tokenizing a given regular expression.

For example,

`(ab|c)*` would be tokenized as **lpar** **char** **char** **or** **char** **rpar** **star**

Write a context-free grammar for token streams of valid regular expressions. Your grammar should be unambiguous, and respect the precedence of the operators.

S
C
A

A ::= C or A
| C

C ::= C S
| S

S ::= S star
| P

P ::= char | lpar A rpar

Q5

Question 5 (10 Points)

Imagine the following is the rules section of a Flex spec:

```
by           {std::cout << "1\n"; }
cr(o+)[^m]   {std::cout << "2\n"; }
[.\n]+       {std::cout << "3\n"; }
...          {std::cout << "4\n"; }
```

What does this spec print on the following input: `bycrom`?

4

4