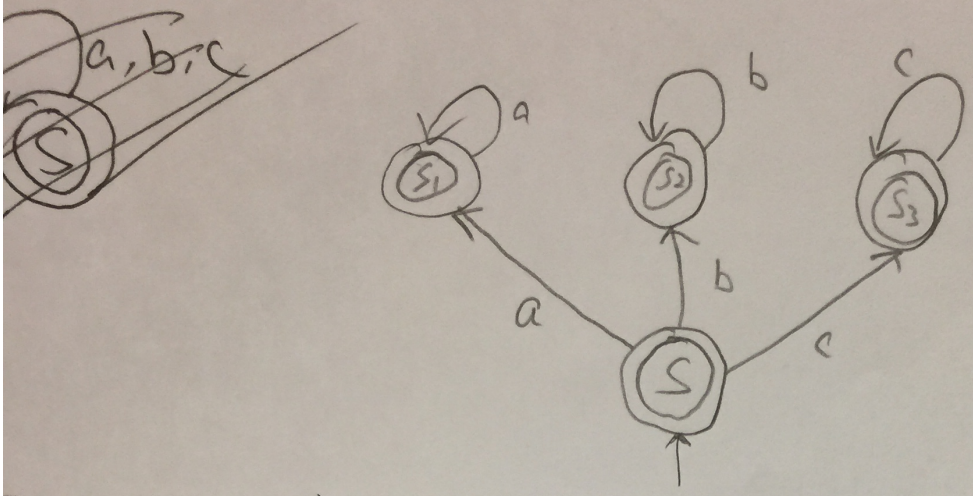
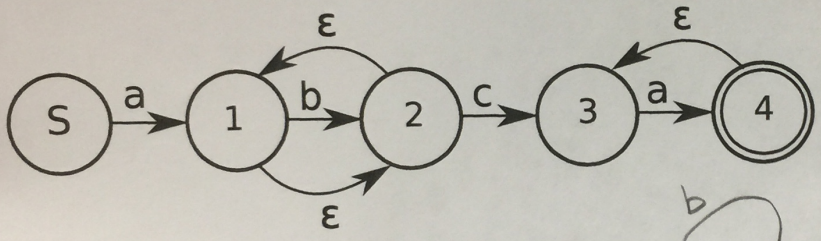
responding to the regular expression (a*|b*|c*)

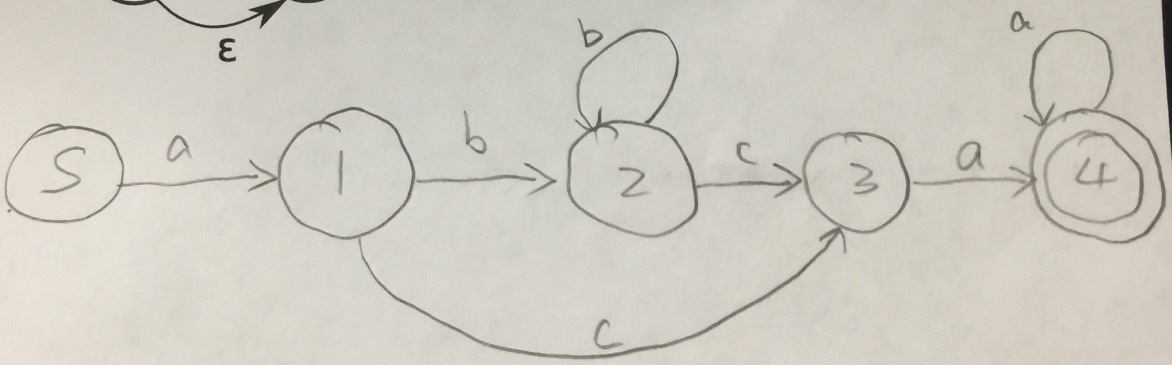


S: start states.

N 2 (3 Points)

NFA with no epsilon transitions that is equivalent to this NFA:

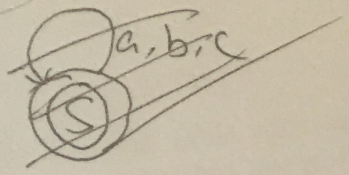


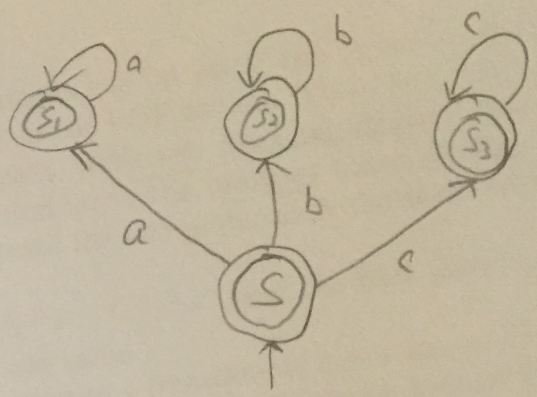


UESTION 1 (3 POINTS)

3

Draw a DFA corresponding to the regular expression (a*|b*|c*)



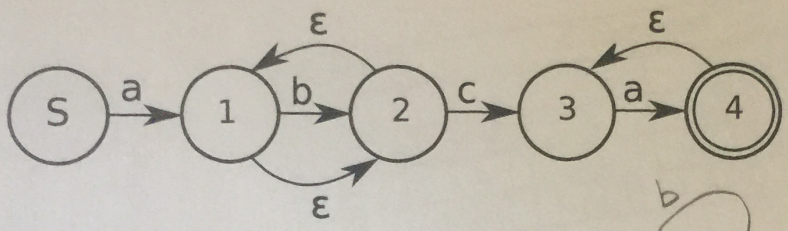


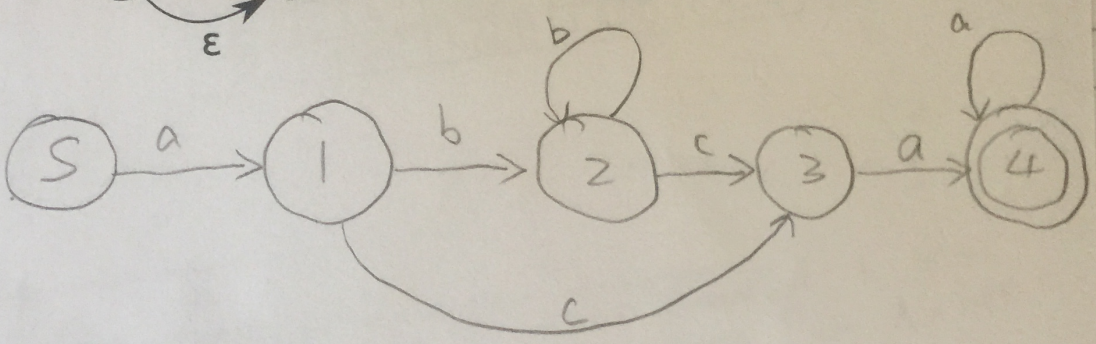
S: start states.

UESTION 2 (3 POINTS)

Draw an NFA with no epsilon transitions that is equivalent to this NFA:

3





returns! Never

QUESTION 3 (4 POINTS)

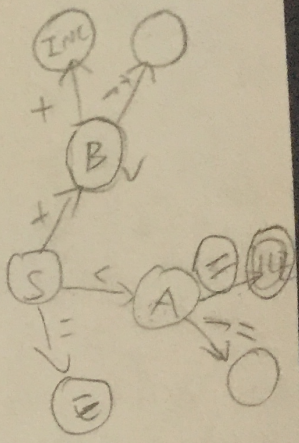
The following state transition table with actions has (at least) two incorrect cells (corresponding to two distinct issues) that are **not** in the EOF column. These flaws will cause the tokenizer to produce an incorrect token stream. Write two character streams, each of which exercises one or the other incorrect cells, and write the incorrect token streams that will be produced. An example is provided, indicate the two problems in the same fashion below that.

Note that the alphabet contains ONLY "+", "<", "=" (and the EOF pseudocharacter): there is no need for an "other" column in the table.

Hint: *Cells containing a single character refers to a transition to a new row. Cells containing a pair refer to the number of characters to rewind ("spit back") into the character stream, and the token to return, respectively.*

Token	Characters
INC	++
ADD	+
LE	<=
L	<
E	=

State	Character			
	+	<	=	EOF
S	0,ADD	A	0,E ✓	S
A	1,L	1,L	1,LE ✗	2,L
B	0,INC	1,ADD	1,ADD	0,ADD



EXAMPLE

Character Stream: <

Corresponding Incorrect Token Stream: L L L ...

(Valid character streams ending in < cause an infinite stream of L tokens because the tokenizer spits back 2 characters and returns L in A x EOF

1st PROBLEM

Example Character Stream: < = EQ

Corresponding Incorrect Token Stream: LE LE LE ...

Valid character streams ending in = cause an infinite steam of LE, because <= can decide end, do not need one put back!

2nd PROBLEM

Example Character Stream: + ++

Corresponding Incorrect Token Stream: ++ Add Add

two plus is increment, but when S read first plus, it returns! Never reach increment (++),

-1 Close, but it's not an infinite stream. It actually returns LE E

-0.5 this is the correct logic, but the token stream of a single plus does not exercise it