

Answer all of the following questions.

1. (5 points) Consider these regular expressions in this order:

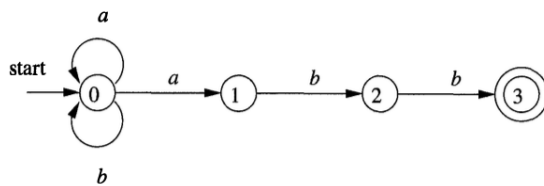
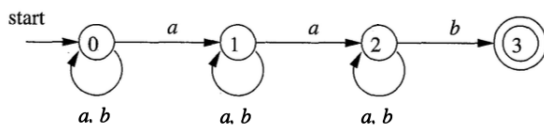
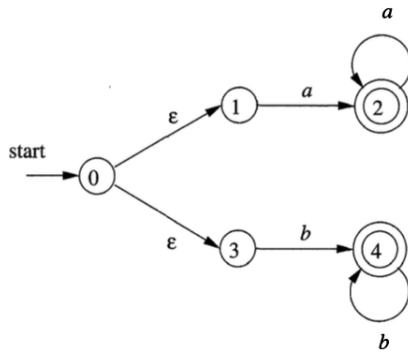
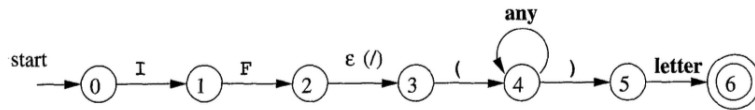
- 1) aac
- 2) ba^+c
- 3) ba^*
- 4) $ccc/.^*ccc$
- 5) $(b|c)^*$
- 6) a^+b
- 7) $(cac|cab)$

Partition the following string into tokens by grouping the characters into lexemes. Mark the lexemes as "Rule 1" through "Rule 7" according to which regex will match them.

• $a\ b\ c\ c\ b\ a\ a\ c\ c\ c\ a\ b\ a\ b\ a\ b\ c\ a\ c\ c\ c\ a\ a\ a\ b\ c\ a\ b\ b\ a\ a\ a\ a\ c\ b\ a\ a\ a$

• $c\ c\ c\ c\ c\ c\ a\ b\ c\ a\ b\ c\ c\ a\ a\ b\ a\ a\ a\ a$

2. (5 points) Fill out the middle portion of a Flex file with the regular expressions that would generate the finite automata below. Add actions in `{ }` braces that prints the contents of the regular expression:



%%

%%

3. (5 points) Consider the following grammar. Parse the string **abcdefg** as Bison would, using an LALR approach. Draw the entire parse tree.

$$S \rightarrow A$$

$$A \rightarrow aBE$$

$$B \rightarrow bCD$$

$$C \rightarrow c$$

$$D \rightarrow d$$

$$E \rightarrow eFG$$

$$F \rightarrow f$$

$$G \rightarrow g$$

4. (5 points) Suppose that you write the following ambiguous grammar in Bison. Draw the entire parse tree on the input $-3-4-5$. Draw every state of the stack from the beginning to the end of the parsing and circle the states that incur a **shift/reduce** conflict in Bison. Finally, write a new, updated grammar that has more levels of precedence (splitting the grammar so that unary minus lives in a higher precedence rule) and therefore does *not* have a shift/reduce conflict. Do not use `%left` or `%right`.

```
%token NUM '-'  
  
%%  
expr:  
    expr '-' expr  
    | '-' expr  
    | NUM  
    ;  
%%
```