Lab Two

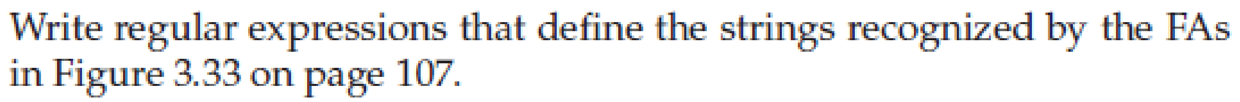
John DeFalco
john.defalco1@marist.edu

September 11th, 2019

1 Crafting a Compiler

Below are the examples listed on the requirements document for the lab from the *Crafting a Compiler* textbook.

1.1 Problem 3.3



Write regular expressions that define the strings recognized by the FAs in Figure 3.33 on page 107.

In order starting with the first FA, the regular expressions would be:

- (ab*a|ba*b)
- a(bcda|cda)?
- (ab*c)?

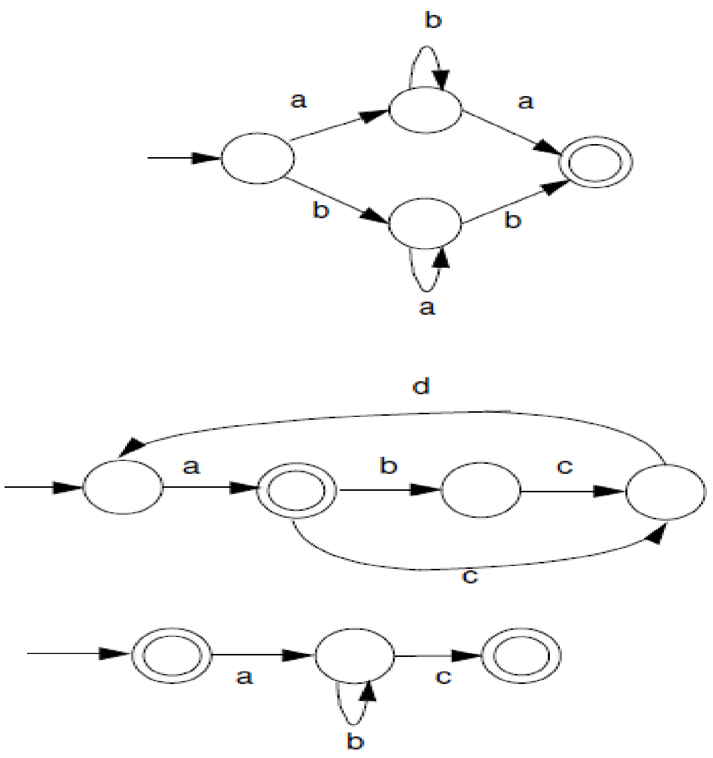
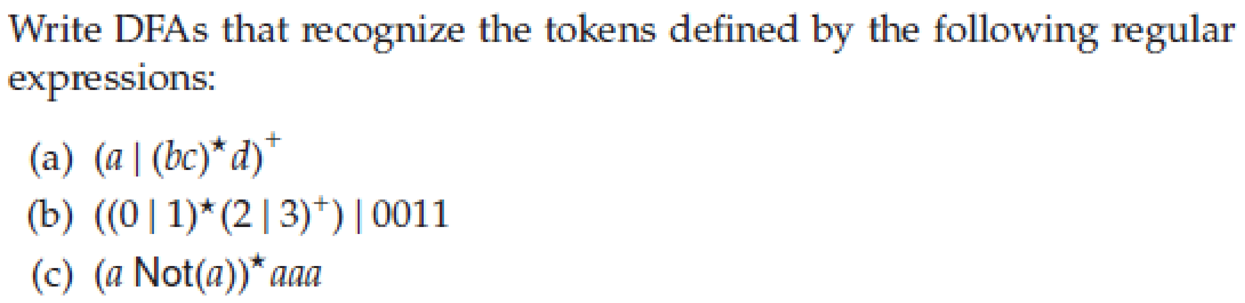


Figure 1.1: The figure mentioned in problem 3.3

1.2 Problem 3.4



Write DFAs that recognize the tokens defined by the following regular expressions:

(a) $(a \mid (bc)^{\star} d)^{+}$

(b) $((0 \mid 1)^{\star}(2 \mid 3)^{+}) \mid 0011$

(c) $(a\ \mathsf{Not}(a))^{\star}\, aaa$

Starting with (a), the DFA, M would be defined in the following 5-tuple:

- $Q = \{q0, q1, q2, q3, q4\}$
- $\Sigma = \{a, b, c, d\}$
- $\delta =$ *See Table 2.1*
- $S = \{q0\}$
- $F = \{q1\}$

Where the state $q4$ represents the error state, if the string tested against the DFA fails to match the original regular expression.

State	Input(a)	Input(b)	Input(c)	Input(d)
q0	q1	q2	q4	q1
q1	q1	q2	q4	q1
q2	q4	q4	q4	q4
q3	q4	q2	q3	q1
q4	q4	q4	q4	q4

Table 1.1: Transition function for 3.4(a)

For the regular expression (b), the DFA N is defined as follows:

- $Q = \{q0, q1, q2, q3, q4, q5, q6, q7\}$
- $\Sigma = \{0, 1, 2, 3\}$
- $\delta =$ *See Table 2.2*
- $S = q0$
- $F = q3$

With $q7$ representing the error state.

State	Input(0)	Input(1)	Input(2)	Input(3)
q0	q1	q2	q3	q3
q1	q4	q2	q7	q7
q2	q2	q2	q3	q3
q3	q7	q7	q3	q3
q4	q2	q5	q7	q7
q5	q2	q6	q7	q7
q6	q2	q2	q7	q7
q7	q7	q7	q7	q7

Table 1.2: Transition function for 3.4(b)

Finally, for the regular expression (c), the DFA O is defined as follows:

- $Q = \{q0, q1, q2, q3, q4, q5\}$
- $\Sigma = \{a, Not(a)\}$
- $\delta =$ *See Table 2.3*
- $S = q0$
- $F = q5$

With representing the error state.

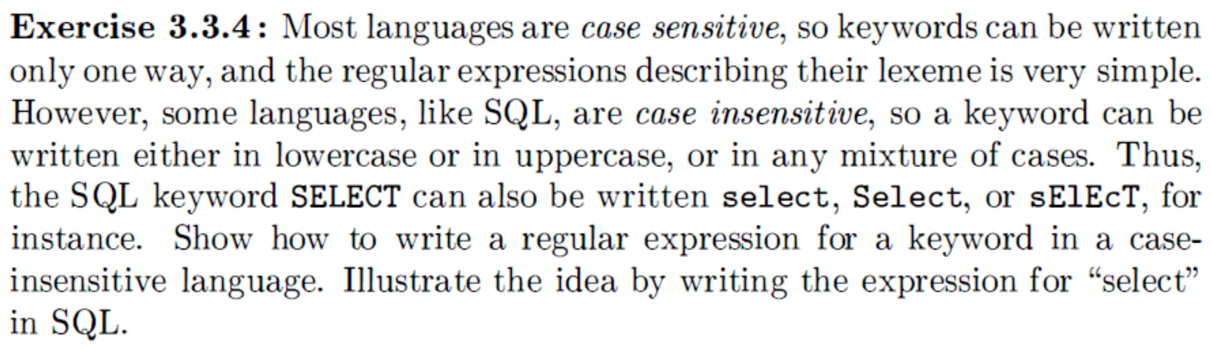
State	Input(a)	Input(Not(a))
q0	q1	q5
q1	q2	q4
q2	q3	q5
q3	q5	q5
q4	q1	q5
q5	q5	q5

Table 1.3: Transition function for 3.4(b)

2 Dragon Book

Below is the example listed on the requirements document for the lab from the *Compilers (Dragon)* textbook .

2.1 Problem 3.3.4



Exercise 3.3.4 : Most languages are *case sensitive*, so keywords can be written only one way, and the regular expressions describing their lexeme is very simple. However, some languages, like SQL, are *case insensitive*, so a keyword can be written either in lowercase or in uppercase, or in any mixture of cases. Thus, the SQL keyword `SELECT` can also be written `select`, `Select`, or `sElEcT`, for instance. Show how to write a regular expression for a keyword in a case-insensitive language. Illustrate the idea by writing the expression for "select" in SQL.

To write an expression that accepts the word SELECT, in upper or lower case, all we would need it the | operator. Using the | operator will tell the scanner that either the lower case or upper case version of the specified letter within SELECT can be accepted. We can make pairs in parenthesis for each character, one being upper case and one being lower case, separated by the | to make a regex that is case-insensitive with the word SELECT. The regex that the problem is calling for will end up being:

- (s|S)(e|E)(l|L)(e|E)(c|C)(t|T)