# Lab Two

Alex Smith

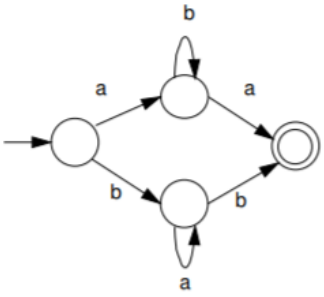
alex.smith1@Marist.edu

February 8, 2019

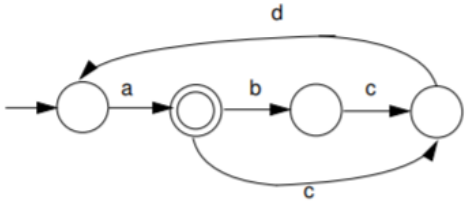
## 1 Crafting a Compiler

### 1.1 Exercise 3.3

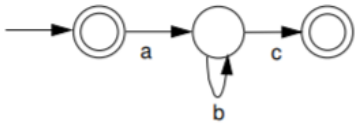
Write regular expressions that define the strings recognized by the FAs in Figure 3.33 on page 107.




This DFA has the alphabet (a,b) and requires that the starting symbol is the same as the ending symbol. When this is expressed as a regular expression, it is:
$((ab^*a)|(ba^*b))$




This DFA has the alphabet (a,b,c,d) and ends with an a. It can be optionally followed by either bcda or cda zero or more times. Written as a regular expression, it is:
$(a((bc)|cda)^*)$

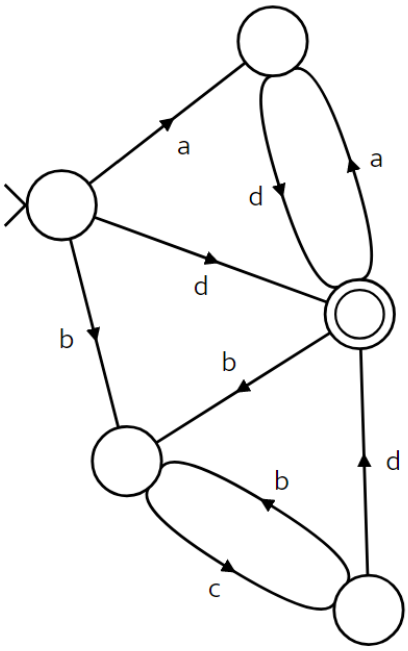



This DFA has the alphabet (a,b,c) and is either empty or starts with an a and ends with a c with zero or more bs between the a and the c. Written as a regular expression, it is:
$\epsilon|(ab^*c)$

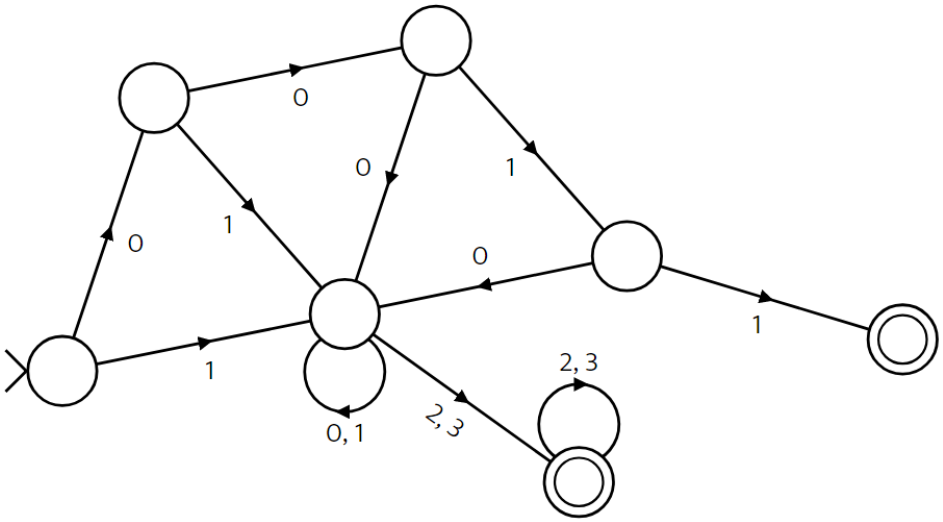
## 1.2 Exercise 3.4

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

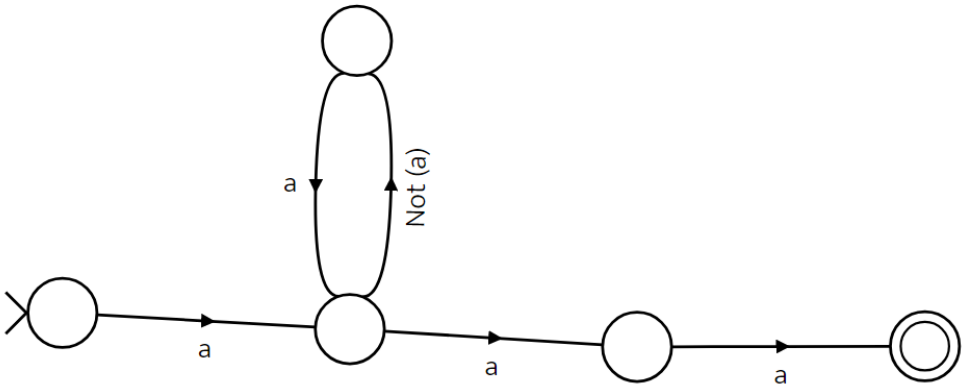
a) $(a|(bc)^*d)^+$



b) $((0|1)^*(2|3)^+)|0011$

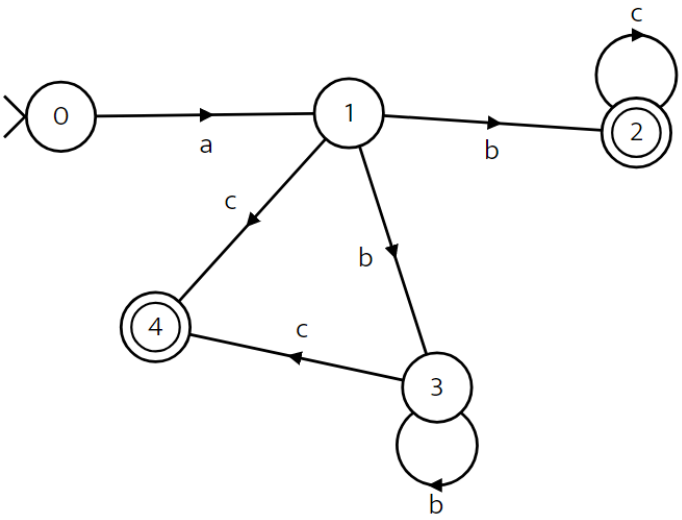



c) $(aNot(a))^*aaa$



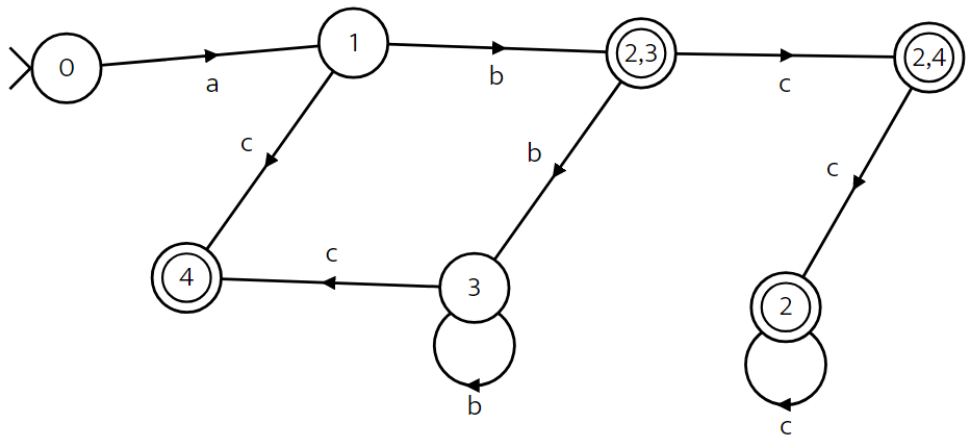
## 1.3 Exercise 3.15

Show the NFA that would be created for the following expression using the techniques of Section 3.8:
$(ab^*c)|(abc^*)$

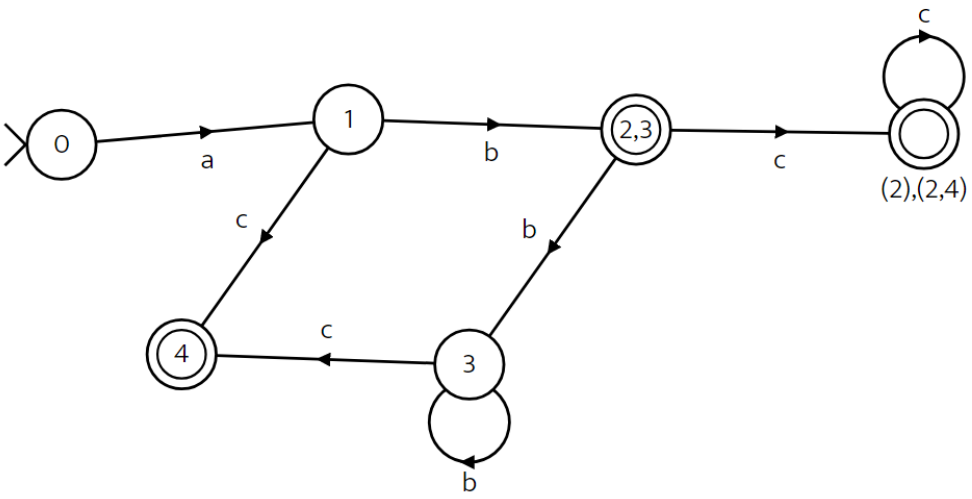



Using MakeDeterministic, translate the NFA into a DFA.
* in the state column denotes an accepting state

| Transition Table | | | |
|---|---|---|---|
| | a | b | c |
| 0 | 1 | | |
| 1 | | 2,3 | 4 |
| 2,3* | | 3 | 2,4 |
| 2,4* | | | 2 |
| 2* | | | 2 |
| 3 | | 3 | 4 |
| 4* | | | |




Using the techniques of Section 3.8.3, optimize the DFA you created into a minimal state equivalent.




The DFAs and NFAs created in Exercise 3.4 and 3.15 were made using:

http://homepages.inf.ed.ac.uk/s1020995/fsmworkbench/create.html

This is an open source FSM workbench created by Matthew Hepburn which is available in browser or on GitHub with the source code.

# 2 The Dragon Book

## 2.1 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 caseinsensitive language. Illustrate the idea by writing the expression for "select" in SQL.

Uppercase and lowercase letters are treated as separate symbols by the lexer so in order to include both as a valid symbol, the 'or' connection must be used for each letter. For example, the regular expression for "select" in SQL would be:
( (s|S) (e|E) (l|L) (e|E) (c|C) (t|T) )