

CMSC330 Spring 2009 Midterm #1

Name _____

Discussion Time (circle one): 9am 10am

Do not start this exam until you are told to do so!

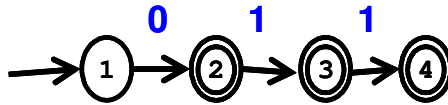
Instructions

- You have 50 minutes for to take this midterm.
- This exam has a total of 100 points, so allocate 30 seconds for each point.
- This is a closed book exam. No notes or other aids are allowed.
- If you have a question, please raise your hand and wait for the instructor.
- Answer essay questions concisely using 2-3 sentences. Longer answers are not necessary and a penalty may be applied.
- In order to be eligible for partial credit, show all of your work and clearly indicate your answers.
- Write neatly. Credit cannot be given for illegible answers.

	Problem	Score
1	REs & finite automata	/8
2	RE to NFA	/12
3	NFA to DFA	/16
4	DFA minimization	/16
5	Programming languages	/6
6	Ruby features	/12
7	Ruby programming	/30
	Extra credit	/10
	Total	/100

1. (8 pts) Regular expressions and finite automata
 - a. (4 pts) Explain what we are referring to when we say regular expressions and finite automata are equivalent.

- b. (4 pts) Give a regular expression equivalent to the following finite automaton.

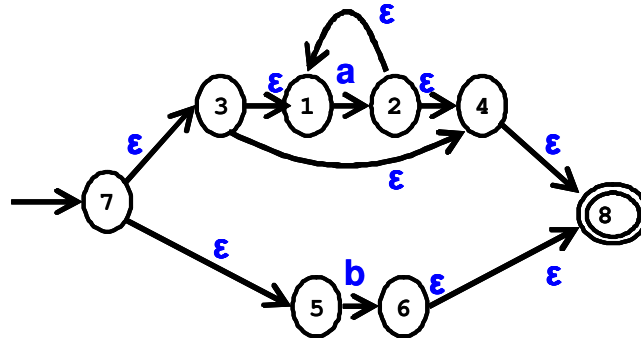


2. (12 pts) Regular expression to NFA

Create a NFA for the regular expression $a(bc)^*$ using construction method from class.

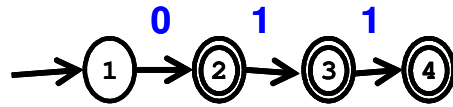
3. (16 pts) NFA to DFA

Apply the subset construction algorithm to convert the resulting NFA to a DFA. Show the NFA states associated with each state in your DFA.



4. (16 pts) DFA Minimization

Apply Hopcroft minimization to the following DFA to generate a minimized DFA. List the DFA states in each partition created, and explain why the partition will or will not be split by the minimization algorithm, and list the DFA states in each resulting partition if the partition is split. You must show each partition created and explain why it is or is not split.



5. (6 pts) Programming languages

- a. Name a disadvantage of dynamic types in terms of program correctness. Provide a code example in Ruby showing the disadvantage.

6. (12 pts) Ruby features

What is the output (if any) of the following Ruby programs? Write FAIL if code does not execute.

a. `a = []` // Output =
`a[1] = "b"`
`a[2] = 3`
`a.each{ |x| puts x }`

b. `a = []` // Output =
`a[1] = "b"`
`a["2"] = 3`
`a.each{ |x| puts x }`

c. `a = { }` // Output =
`a[1] = "b"`
`a["2"] = 3`
`a.each{ |x,y| puts x }`

d. `if ("terps rule!" =~ /[a-z]+)/` // Output =
`puts "#{ $1 }"`
`puts $2`
`else`
`puts "None"`
`end`

7. (30 pts) Ruby programming

Consider the following programming problem. Suppose you want to sort the list of alphabet symbols in output file of the Finite Automata (FA) project. Your Ruby program should read each line of in the file containing the output. For lines that represent a list of symbols in the alphabet of the FA, you should print the symbols in alphabetical order. Any other line you should leave alone and print the line as is.

Write a complete Ruby program that performs this sorting. The format of the output file of the FA project is shown in the following example. The portions of the output file in boldface are always fixed. Only the non-boldface portions of the FA description may change. You may assume that all symbols in the alphabet are lowercase letters between a and z, and that they are always preceded by a single space in the output file.

For (10 pts) extra credit, you may also have your program print the list of states & final states in numerical order. You may assume that all states are numbers ≥ 0 .

Helpful functions: `File.new(filename, mode)` // opens filename in mode, returns File
`file.eof?` // is File object file at end?
`file.readline` // read single line from file
`str.scan(...)` // finds patterns in String str
`str.to_i` // returns int value for String str
`print(...)` // prints arguments without newline

Input	Output	Extra Credit Output
<code>% Start 9</code>	<code>% Start 9</code>	<code>% Start 9</code>
<code>% Final { 10 9 }</code>	<code>% Final { 10 9 }</code>	<code>% Final { 9 10 }</code>
<code>% States { 11 9 10 }</code>	<code>% States { 11 9 10 }</code>	<code>% States { 9 10 11 }</code>
<code>% Alphabet { b a }</code>	<code>% Alphabet { a b }</code>	<code>% Alphabet { a b }</code>
<code>% Transitions {</code>	<code>% Transitions {</code>	<code>% Transitions {</code>
<code>% (9 a 11)</code>	<code>% (9 a 11)</code>	<code>% (9 a 11)</code>
<code> ...</code>	<code> ...</code>	<code> ...</code>
<code>% (9 10)</code>	<code>% (9 10)</code>	<code>% (9 10)</code>
<code>% }</code>	<code>% }</code>	<code>% }</code>

