CMSC330 Fall 2014 Midterm 1 Grading

1. (8 pts) Programming languages (PL) et al. For the following multiple choice questions, circle the letter(s) on the right corresponding to the best answer(s) to each question.

   a. Which following PL feature(s) is *not* a part of Ruby? A **B** C D
      A) object-oriented programming
      B) method overloading
      C) implicit declarations
      D) dynamic types

   b. Which value(s) in the guard of an Ruby if statement cause the then branch to be executed? E.g., if (guard) then x else y end **A** B C **D**

      A) true B) false C) nil D) 0

   c. Which Ruby feature(s) makes "undefined variable" errors more likely? A B **C** D
      A) code blocks
      B) regular expressions
      C) implicit declarations
      D) dynamic types

   d. Ruby code blocks are an example of which PL feature(s)? A B **C** D
      A) object-oriented programming
      B) functional programming
      C) higher-order functions
      D) imperative programming

   e. For which PL(s) is expression "x == y" *not* testing physical equality? A B **C** D

      A) Java B) C C) Ruby D) OCaml

   f. Which following operation(s) is *not* essential for regular expressions? **A** B C D
      A) + (e.g., a+)
      B) * (e.g., a*)
      C) . (e.g., ab)
      D) | (e.g., a|b)

   g. Which following feature(s) is *not* allowed in an NFA? A B C **D**
      A) state with multiple transitions for the same label
      B) epsilon transitions
      C) multiple final states
      D) multiple start states

   h. Which following feature(s) is *not* allowed in an OCaml expression? A **B** C D
      A) lists with different numbers of elements
      B) lists with different types of elements
      A) tuples with different numbers of elements
      B) tuples with different types of elements

2. (10 pts) Ruby. What is the output (if any) of the following Ruby programs? Write FAIL if code does not execute. Output "nil" for "puts x" when x is nil (as in Ruby 1.8.7), instead of outputting a blank line (as in Ruby 1.9.3).

a.
```
a = 1
puts a+1
a = "terp"
puts a
```
OUTPUT = **2**
terp

b.
```
a = [ ]
a[1][2] = 3
puts (4 + a[1][2])
```
OUTPUT = **FAIL**

c.
```
a = [ ]
a["terp"] = "tesudo"
a[5] = 6
a.each { |x| puts x }
```
OUTPUT = **FAIL**

d.
```
"Univ of Maryland" =~ /([a-z]+)/
puts "Found #{$1}" if $1
puts "Found #{$2}" if $2
puts "Found #{$3}" if $3
```
OUTPUT = **Found niv**

e.
```
a = "Univ of Maryland".scan(/[a-z]+/)
a.each { |x| puts "Found #{x}" }
```
OUTPUT = **Found niv**
Found of
Found aryland

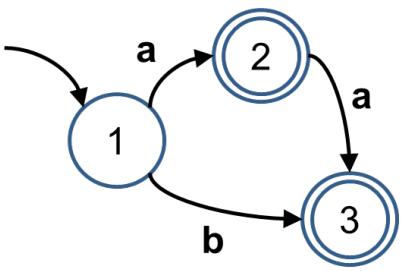
3. (6 pts) Regular expressions and finite automata.

a. (2 pts) Write a regular expression for all *even* integers between 10 and 999.

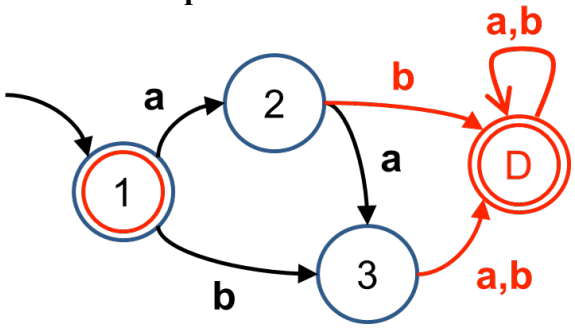
(1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9|epsilon)(0|2|4|6|8)
OR
(1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)(0|2|4|6|8) | (1|2|3|4|5|6|7|8|9)(0|2|4|6|8)
OR
/^[1-9][0-9]?[02468]$/

b. (4 pts) Create a DFA that accepts a string (composed of a's and b's) if and only if it is NOT accepted by the following DFA.

Original

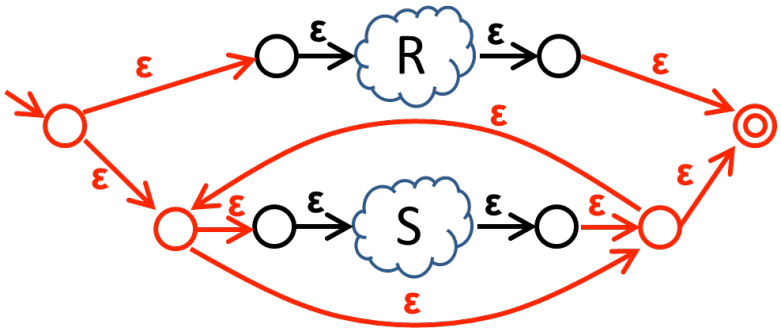


Complement



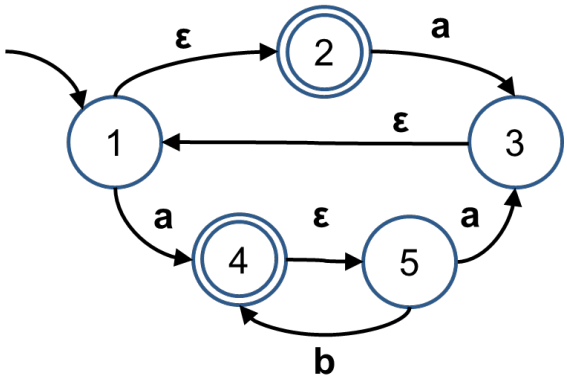
4 pts all correct
2 pts partially correct

4. (4 pts) RE to NFA. Create a NFA for the regular expression **R|(S*)** given the NFAs for **R** & **S** below. You *must* use the method described in lecture. Note that **R** and **S** are regular expressions, not symbols in the alphabet.



4 pts all correct
2 pts partially correct

5. (14 pts) NFA to DFA. Consider the NFA given on the right:

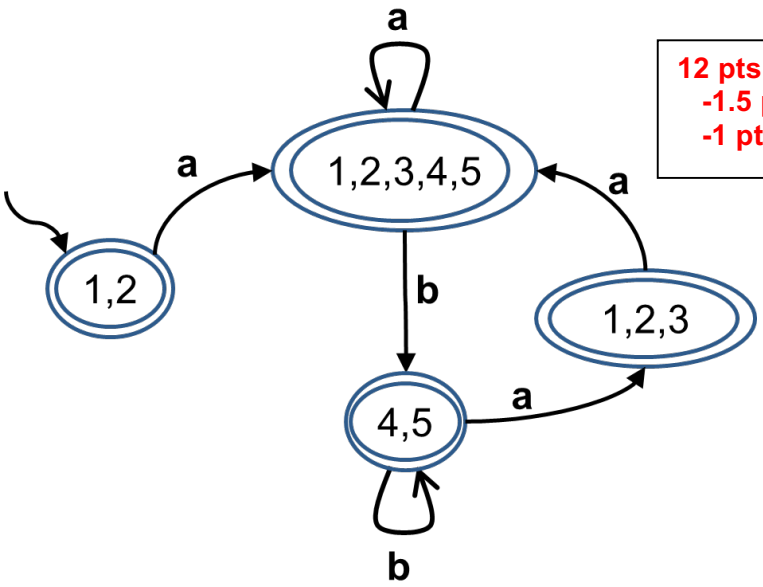


a. (2 pts) Does the NFA accept the string "aab"? If it accepts the string, list a sequence of state transitions (e.g., 1,2,3) that leads to acceptance.

2 pts all correct

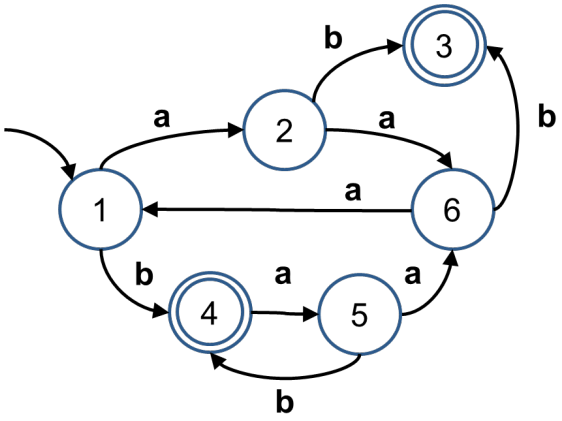
Yes. 1,2,3,1,4,5,4

b. (12 pts) Reduce the NFA to a DFA using the subset construction algorithm discussed in class. Be sure to label each state in the DFA with the corresponding state(s) in the NFA.



12 pts
-1.5 pts each wrong state
-1 pt each wrong transition

6. (6 pts) DFA Minimization. Consider applying the Hopcroft DFA minimization algorithm discussed in class to the following DFA.



a. (2 pts) What are the initial partition(s) created by the Hopcroft algorithm?

P1 { 1,2,5,6 }, P2 { 3,4 }

b. (2 pts) Do any partitions need to be split? If yes, what are the resulting partitions after the split?

P2 needs to be split. Result = P1 { 1,2,5,6 }, P2 { 3 }, P3 { 4 }

c. (2 pts) Is the DFA minimization algorithm finished at this point? Explain.

Not finished, since P1 now needs to be split.

7. (40 pts) Ruby programming

Implement a Ruby program in roster.rb that reads a student database file where each line contains a student UID, student name, college, and major. Your program must process this file and display two lists. The first list displays each college (sorted by college name). For each college, display on a separate line the college name, the number of students in the college, and each student in the college (sorted by student name). The second list displays each student (sorted by student name). For each student, display on a separate line the student's UID, student name, and majors (sorted by major name). Your program will be called with the name of the database file as an argument. For example, running your program on a file called data.txt ("ruby roster.rb data.txt") could generate:

```
% more data.txt
0002: Cathy,ENGR,Computer Engineering
0004: David,BSOS,Economics
0003: Bob,BSOS,Psychology
0008: Alice,CMNS,Computer Science
0004: Bob,BSO,Economics
0008: Alice,CMNS,Biology
0002: Cathy,CMNS,Math
0005: Ellen,CMNS,Computer Science
0002: Cathy,CMNS

% ruby roster.rb data.txt
ERROR 0004: Bob,BSO,Economics
ERROR 0002: Cathy,CMNS
COLLEGES
BSOS,2,Bob,David
CMNS,3,Alice,Cathy,Ellen
ENGR,1,Cathy
STUDENTS
0008,Alice,Biology,Computer Science
0003,Bob,Psychology
0002,Cathy,Computer Engineering,Math
0004,David,Economics
0005,Ellen,Computer Science
```

Helpful Functions	
f = File.new(n, mode)	// opens n in mode, returns File f
f.eof?	// is File object f at end?
ln = f.readline	// read single line from file f into String ln
a = f.readlines	// read all lines from file into array a
a = str.scan(…)	// finds patterns in String str, returns in array a
a = h.keys	// returns keys in hash h as an array a
a.sort	// returns sorted version of array a
a.sort { |x,y| ... }	// sort using code block as comparator
a.sort!	// sorts elements of array a in place
a.size	// number of elements in the array
a.join(x)	// create str by joining array elements separated by str x
a.each { … }	// apply code block to each element in array
a.push / a.pop	// treat array as stack
s.to_i	// convert string s to int value
ARGV	// array containing command line arguments

Student UIDs must be 1 or more digits, and are separated from student names by a colon and a space. Student names must be composed of 1 or more lowercase and uppercase characters. College names must be exactly four uppercase letters. Majors must be composed of 1 or more lowercase and uppercase letters, and may include spaces. College names and majors are separated by commas. Lines that do not follow this format should produce an error message and otherwise be ignored.

Students may have an arbitrary number of majors (e.g., Alice, Cathy). Each student may only be counted once for each college, even if they have multiple majors within the college (e.g., Alice only counts as 1 student for CMNS).You may assume each student has a single unique UID. While reading in the file, for each invalid line found, your program should output ERROR followed by the invalid line. Next, it should output "COLLEGES", followed by the information for each college (in sorted order by college name). Finally, it should output "STUDENTS", followed by the information for each student (in sorted order by student name).

Example Solution:

```
f = File.new(ARGV[0], "r")
lines = f.readlines
uid = { }
student = { }
college = { }

lines.each { |line|
  if line =~ /^([0-9]+): ([A-Za-z]+),([A-Z]{4}),([A-Za-z ]+)$/
       uid[$2] = $1
       student[$2] = [ ] if !student[$2]
       student[$2].push $4
       college[$3] = { } if !college[$3]
       college[$3][$2] = true
  else
       puts "ERROR #{line}"
  end
}

puts "COLLEGES"
college.keys.sort.each { |c|
   puts "#{c},#{college[c].size},#{college[c].keys.sort.join(",")}"
}

puts "STUDENTS"
student.keys.sort.each { |s|
   puts "#{uid[s]},#{s},#{student[s].sort.join(",")}"
}
```

40 pts all correct

6 pts Initialize
- 2 pts Command line
- 2 pts I/O
- 2 pts iterate over lines

10 pts Regular expression
- 6 pts extract & store info
- 4 pts ensure correct format

14 pts Data structures
- 4 pts store student info
- 4 pts store college info
- 4 pts multiple majors for student
- 2 pts duplicate student for college

10 pts Output
- 2 pts output ERROR
- 4 pts output colleges
- 4 pts output students

8. (12 pts) OCaml Types and Type Inference

   a. (4 pts) Give the type of the following OCaml expressions

      i. (2 pts) [[2 + 3]] **Type = int list list**
      ii. (2 pts) fun x -> [3] **Type = 'a -> int list**

   b. (4 pts) Write an OCaml expression with the following type

      i. (2 pts) ((int list) * int) list **Code = [([2] , 2)] OR [[2] , 2]**
      ii. (2 pts) 'a list -> 'a -> 'a list **Code = fun x y -> y::x OR let f x y = y::x**

   c. (4 pts) Give the value of the following OCaml expressions. If an error exists, describe the error.

      i. (2 pts) let x = 3 in let x = x+2 in x+1 **Value / Error = 6**
      ii. (2 pts) (fun a -> match a with (x::y::z) -> z) [2 ; 3 ; 4] **Value / Error = [4]**