CMSC330 Spring 2012 Midterm #1 Solutions

1. (6 pts) Programming languages
   a. (3 pts) Explain how scripting languages like Ruby have different goals than object-oriented languages like Java.

      Quick short programs vs. large, long-lived, easily maintained programs.

   b. (3 pts) Based on (a), explain why Ruby uses implicit variable declarations, but Java does not.

      Implicit declarations make programming quicker, but may lead to more errors.

2. (6 pts) Ruby
   What is the output (if any) of the following Ruby programs? Write FAIL if code does not execute.

   a. (3 pts)

```
a = [ ]
a[1] = 0
puts "foo" if a[1]
puts a[2]
```

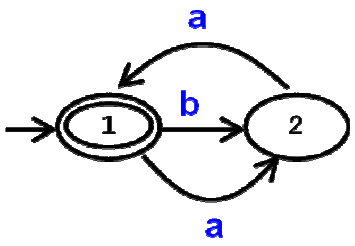
   # Output = **foo**
   nil

   b. (3 pts)

```
a = { }
a[1] = 0
puts a["1"]
puts a[0]
```

   # Output = **nil**
   nil

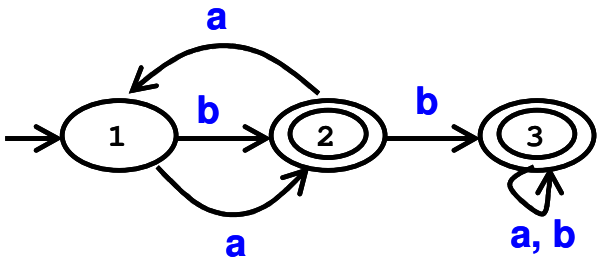
3. (10 pts) Regular expressions and finite automata. Consider the following DFA:



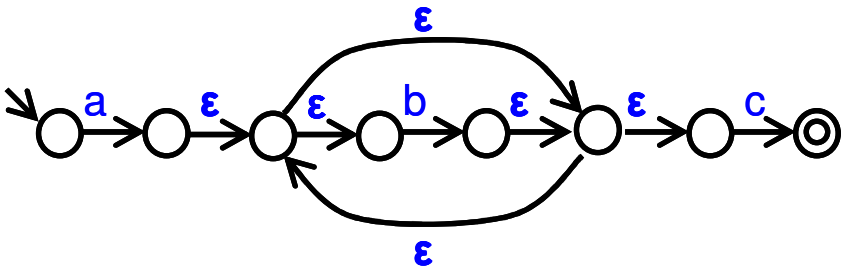
a. (4 pts) Give a regular expression for the strings accepted by the DFA. Use only the concatenate, union, and closure operations. I.e., do not use Ruby regular expressions.

RE = (ba|aa)* or ((b|a)a)*

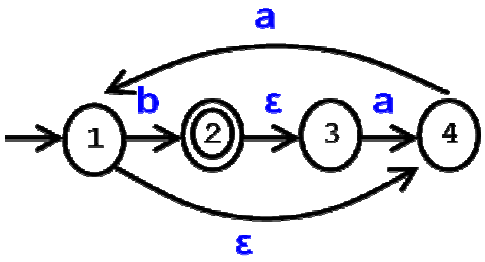
b. (6 pts) Give a DFA that accepts a string if and only if it is not accepted by the DFA above.



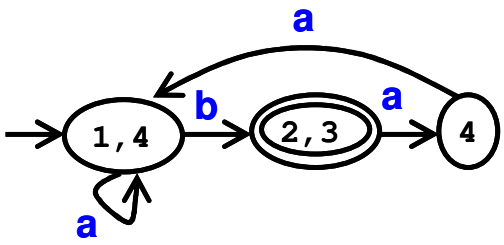
4. (8 pts) RE to NFA
Create a NFA for the regular expression ab*c using the method described in lecture.



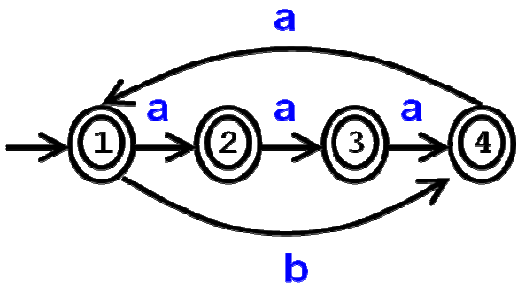
5. (16 pts) NFA to DFA
Apply the subset construction algorithm discussed in class to convert the following NFA to a DFA. Show the NFA states associated with each state in your DFA.



Answer:



6. (10 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?

{ 1,2,3,4 }

b. (4 pts) Do any partitions need to be split? If yes, what is the result after splitting the partition?

YES
{ 1 }, { 2,3,4 }

c. (4 pts) Is the DFA minimization algorithm finished at this point? Explain.

NO
Partition { 2,3,4 } will need to be split since 4 behaves differently for a.

7. (28 pts) Ruby programming

Consider the FiniteAutomaton project, where you wrote code to implement finite automata. Recall that the code in 06-dfa.rb used a hash of hashes to store transitions in a finite automaton. For instance, transition = { }, transition[1] = { }, transition[1]["a"] = 2 was used to store the transition from state 1 to 2 for symbol "a".

Your goal is to write Ruby code to 1) read in a file containing a list of transitions in a DFA, 2) reject any invalid transitions, 3) put the list of transitions into the *transition* hash, and 4) use the *transition* hash to print out the valid transitions in sorted order. Valid transitions will be in the form "state symbol state" (e.g., "4 a 7").

You must print out the list of transitions in order using the *transition* hash. You may not simply sort the list of transitions as strings.

For the purpose of this analysis you may assume that all states are integers between 0 and 9, and symbols are lowercase letters. A single space separates the states and the symbol. Your program should accept the name of the text file containing the transitions as a command line argument. It should output "READ", followed by each transitions read or "INVALID" for transitions not in a legal format. It should then output "SORT", followed by all the transitions in lexicographically sorted order (sorted first by starting state, then by transition symbol). Since the finite automaton is a DFA, there will only be one destination for each starting state & transition symbol (i.e., there is no need to sort based on the destination state).

For instance, given the following inputs in text file "foo" when executed as "ruby fa.rb foo" the output of your program should be as follows:

Example 1		Example 2	
Input	Output	Input	Output
0 a 1	READ	0 a 1	READ
2 b 1	0 a 1	2 b 1	0 a 1
2 a 2	2 b 1	2 0 2	2 b 1
1 a 3	2 a 2	1 a 3	INVALID
1 b 0	1 a 3	1 b a	1 a 3
	1 b 0		INVALID
	SORT		SORT
	0 a 1		0 a 1
	1 a 3		1 a 3
	1 b 0		2 b 1
	2 a 2		
	2 b 1		

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!	// sorts elements of array a in place
a.size	// number of elements in the array
a.each { … }	// apply code block to each element in array
a.push / a.pop	// treat array as stack
ARGV	// array containing command line arguments

Sample answer:

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

puts "READ"
lines.each { |line|
   if line =~ /^(\d) ([a-z]) (\d)$/
        trans[$1] = { } if trans[$1] == nil
        trans[$1][$2] = $3
        puts line
   else
        puts "INVALID"
   end
}

puts "SORT"
src = trans.keys.sort
src.each { |x|
   syms = trans[x].keys.sort
   syms.each { |y|
        puts "#{x} #{y} #{trans[x][y]}"
   }
}
```

8. (16 pts) OCaml

   a. (2 pts each) Give the type of the following OCaml expressions.

      i. [[4 ; 3]; [2]] **Type = int list list**
      ii. let f x = [x + 2] **Type = int -> int list**

   b. (3 pts each) Give the value of the following OCaml expressions. If an error exists, describe it.

      iii. [3;2]::[1] **Value = error, not int list list**
      iv. let f x = (match x with h::t -> t) in (f [8;2;4]) **Value = [2;4]**

   c. (3 pts each) Write an OCaml expression with the following type.

      v. int -> int list **Code = let f x -> [x+1]**
      vi. int list -> int **Code = let f (x::y) -> x+1 OR**
      let f z = match z with (x::y) -> x+1