CMSC330 Fall 2010 Midterm #1 Solutions

1. (12 pts) Ruby
   a. (6 pts) List 3 features of Ruby that are often found in scripting languages.

      Examples: interpreted, implicit declarations, dynamic types, built-in regular expressions, easy string processing

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

      i. puts "Found" if "College Park" =~ /([a-z]+)/ **Output = Found ollege**
      puts $1

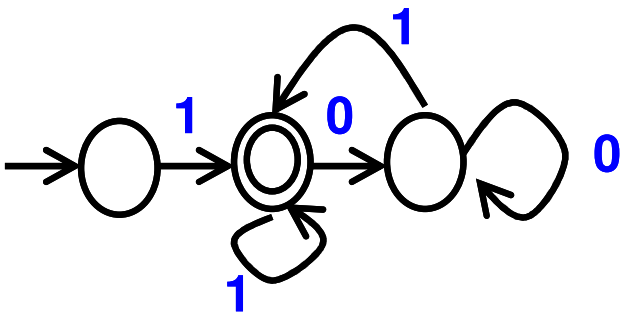
      ii. a = [ ] **Output = nil nil 1**
      a[2] = 1
      puts a

      iii. a = { 1 => 2 } **Output = 2**
      a.keys.each{ |x| puts "#{a[x]}" }

2. (12 pts) Regular expressions and finite automata
   a. (4 pts) Give a regular expression for all binary numbers (strings of 0s and 1s) beginning and ending with 1 (e.g., 1, 111, 10011).

      1 | 1(0|1)*1

   b. (4 pts) Give a DFA for all binary numbers (strings of 0s and 1s) beginning and ending with 1 (including 1).

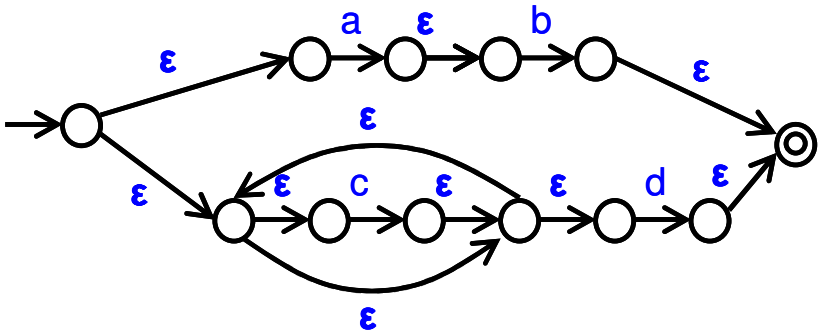


   c. (4 pts) Explain why adding a **not** operator to the standard **union**, **concatenation**, and **closure** operators does not make regular expressions more powerful.

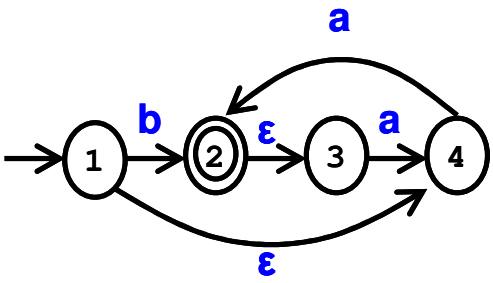
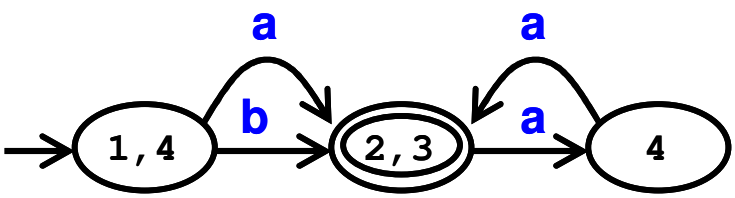
      Because any regular expression written using the not operator can also be written using just the union / concatenation / closure operators.

      [Optional] We know this because any DFA may be converted to the complement-DFA, then converted into a regular expression that does not use the not operator.

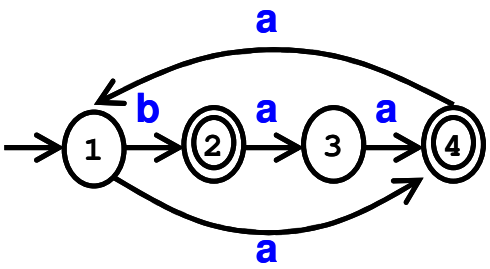
3. (20 pts) RE to NFA to DFA
   a. (8 pts) Create a NFA for the regular expression ab|c*d using the method described in lecture. Remember precedence is in the order: closure > concatenation > union.



   b. (12 pts) 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.

NFA	Resulting DFA
	

4. (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, 3 } and { 2, 4 }

b. (5 pts) Which partition needs to be split first, if any? Explain.

{ 1, 3 } needs to be split first, since state 1 & 3 behave differently for input "b". Until { 1, 3 } is split the nodes 2, 4 in partition { 2, 4 } behave identically and cannot be split.

c. (3 pts) Is the DFA minimization algorithm finished at this point? Explain.

No, since after { 1, 3 } is split, the nodes 2, 4 in partition { 2, 4 } behave differently on input "a" and need to be split.

5. (18 pts) OCaml

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

i. [[1 ; 3] ; [2]] **Type = int list list**

ii. let x y = y + 1 **Type = int -> int**

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

i. [5]::[6] **Error = prepending int list to int list**

ii. let x = 1 in x + 3 **Value = 4**

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

i. int -> int list **Code = let x y = [2 ; y]**

d. (5 pts) Write an OCaml function non_empty that returns true if its argument is a non-empty list and false otherwise. For instance, non_empty [ ] returns false, and non_empty [1] returns true.

```
let non_empty l = match l with
    [] -> false
  | (h::t) -> true
```

6. (28 pts) Ruby programming

Consider the following programming problem. Suppose you want to analyze a text file to determine how many different unique words immediately appear after each word in the file. For the purpose of this analysis you may assume that all words are lowercase or uppercase, ignore all whitespace and punctuation separating words, and assume the last word in the text file does not have any words appearing after it. Your program should accept the name of the text file as a command line argument. It should output a list of words in the text file in sorted order, with the number of different unique words following each word.

For instance, given the text file "foo" when executed as "ruby count.rb foo" the output should be as follows. This has 2 unique words following it (is, test), and test has 2 unique words following it (This, should). All other words only have 1 unique word following them.

Input (content of foo)	Output	Helpful functions

This is a test.
This is a short test.
This test should work.
	This 2
	a 2
	is 1
	short 1
	should 1
	test 2
	work 0
		f = File.new(filename, mode) // opens filename 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! // 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

You may NOT use methods such as include?, has_key?, uniq, join, split, chop, etc.

Sample Solution 1

```
nx = { }
prev = nil
f = File.new(ARGV[0], "r")
lines = f.readlines
lines.each { |line|
  words = line.scan(/[a-zA-Z]+/)
  words.each { |w|
    nx[w] = { } if nx[w] == nil
    nx[prev][w] = true if prev != nil
    prev = w
  }
}
k = nx.keys
k.sort!
k.each { |x| puts "#{x} #{nx[x].size}" }
```

Sample Solution 2

```
nx = { }
prev = nil
f = File.new(ARGV[0], "r")
until f.eof? do
  line = f.readline
  words = line.scan(/[a-zA-Z]+/)
  words.each { |w|
    nx[w] = { } if nx[w] == nil
    nx[prev][w] = true if prev != nil
    prev = w
  }
end
k = nx.keys
k.sort!
k.each { |x| puts "#{x} #{nx[x].size}" }
```