CMSC330 Fall 2011 Midterm #1 Solution

1. (6 pts) Programming languages
   a. (3 pts) List a programming language feature that makes it easier to write programs, but make it more difficult to detect errors in the program. Explain why.

      Implicit declarations (misspelled variable names), dynamic types (type errors), weak typing (type errors), etc…

   b. (3 pts) What language feature in Ruby is similar to anonymous functions in OCaml / Java? Explain.

      Code blocks, since they define unnamed functions that can be passed to methods as arguments.

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

   a. (2 pts)
   ```
   a = "foo"
   if a =~ /(o+)/ then
   puts $1
   end
   ```
   # Output = oo

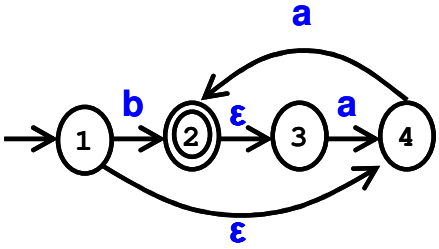
   b. (2 pts)
   ```
   a = [ ]
   a["foo"] = 1
   puts a["foo"]
   ```
   # Output = FAIL

   c. (2 pts)
   ```
   a = { }
   a["foo"] = 1
   puts a["foo"]
   puts a["bar"]
   ```
   # Output = 1
   nil

3. (10 pts) Regular expressions and finite automata

   For this problem, provide regular expressions using only the concatenate, union, and closure operations. I.e., do not use Ruby regular expressions.

   a. (4 pts) Give a regular expression for the set of strings accepted by the following NFA:

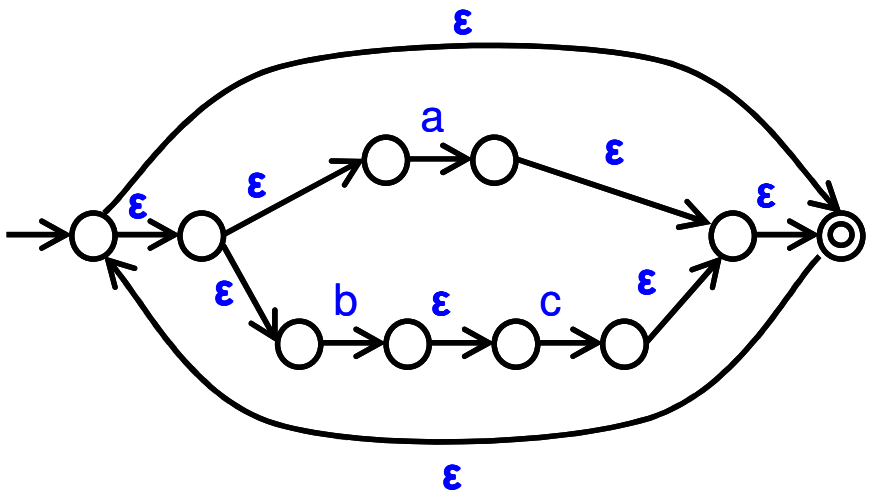


RE = (b|a)(aa)*

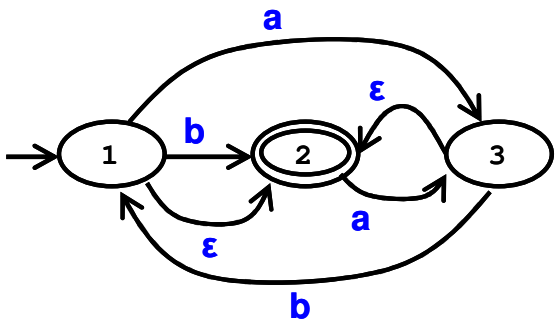
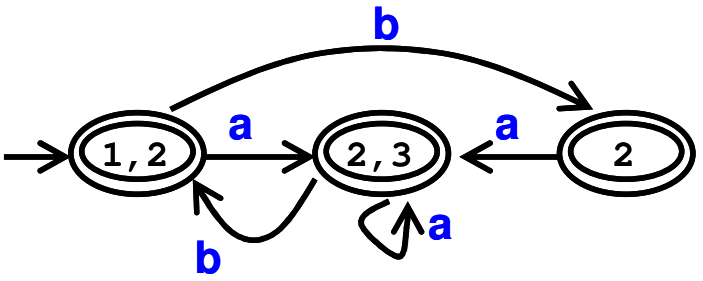
b. (6 pts) Give a regular expression for OCaml expressions of type *int list*. Assume there are no empty lists, and all numbers are binary (i.e., digits consist of 0 or 1). For example, strings generated by your RE include: [1], [100110], [01;0], [0;1;1101], etc…

RE = [(0|1)(0|1)*(;(0|1)(0|1)*)*] or [((0|1)(0|1)*;)*(0|1)(0|1)]

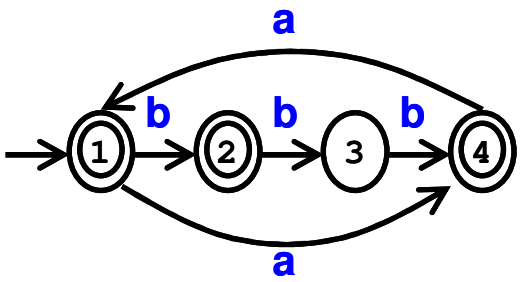
4. (10 pts) RE to NFA
Create a NFA for the regular expression (a|bc)* using the method described in lecture.



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.

NFA	DFA
	

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?

Initial partitions = {1, 2, 4} and {3}

b. (6 pts) Explain why one of the initial partitions needs to be split. What is the result after splitting the partition?

Partition {1, 2, 4} needs to be split since (mention either case below)
- **for input a**
  - **1,4 stays in partition, 2 rejects**
- **for input b**
  - **4 rejects, 1,2 don't reject**
  - **1 stays in partition, 2,4 don't stay in partition**
  - **1 stays in partition, 2 goes to different partition, 4 rejects**

Result of split can be (one of following) respectively

{1,4}, {2}	**// Not yet minimal, since 1,4 can be split**
{1,2}, {4}	**// Not yet minimal, since 1,2 can be split**
{2,4}, {1}	**// Not yet minimal, since 2,4 can be split**
{1,}, {2},{3}	**// Minimal, since all states in own partition**

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

See comment above

7. (24 pts) Ruby programming

Consider the following programming problem. Suppose you want to analyze a text file to determine how many different *unique* **and** *total* 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* **and** *total* words following each word.

Example solution

```
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
    if prev != nil
      nx[prev][w] = 0 if nx[prev][w] == nil
      nx[prev][w] = nx[prev][w]+1
    end
    prev = w
  }
}
k = nx.keys
k.sort!
k.each { |x|
  s = 0
  nx[x].keys.each { |k| s = s + nx[x][k] }
  puts "#{x} #{nx[x].size} #{s}"
}
```

8. (18 pts) OCaml

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

      i. [2 ; 5] **Type = int list**
      ii. let f x = x + 2 **Type = int -> int**

   b. (2 pts each) Give the value of the following OCaml expressions. If an error exists, describe it.

      iii. 3::[2;1] **Value = [3;2;1]**
      iv. let x = 2 in x + 5 **Value = 7**

   c. (2 pts each) Write an OCaml expression with the following type.

      v. int list list **Example Code = [[1;2]]**
      vi. int -> int **Example Code = let f x = x + 2**

   d. (6 pts) Write an OCaml function *singleList* that returns true if its argument is a list with exactly one element and false otherwise. For instance, singleList [4] returns true, and singleList [] and singleList [3;2] both return false.

Example solutions

```
let singleList l = match l with
        [x] -> true
        | _ -> false

let singleList l = match l with
        x::[] -> true
        | _ -> false

let singleList l = match l with
        [x] -> true
        | [] -> false
        | x::y -> false

let singleList l = match l with
        [x] -> true
        | [] -> false
        | _::_ -> false
```