CMSC330 Fall 2009 Midterm #1 Solutions

1. (9 pts) Programming languages
    a. (3 pts) Explain the difference between interpretation and compilation.

        Interpretation evaluates a program one line at a time. Compilation translates a program into a different form that can be executed.

    b. (6 pts) Explain the difference between static and dynamic types. Write a small piece of Ruby code that relies on dynamic types.

        With static types, variable types are checked at compile time and thus generally remain fixed. With dynamic types, variable types are checked at run time (right before they are used), and can thus change during program execution.

        Example: x = 1 ; x = "a" ;

2. (15 pts) Regular expressions and finite automata
    a. (5 pts) Give a regular expression for all binary numbers (strings of 0s and 1s) with an odd number of 0s.

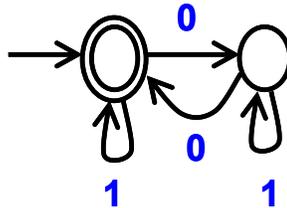   **Many possible answers, some examples are:**

        1* 0 (1*01*0)*1*
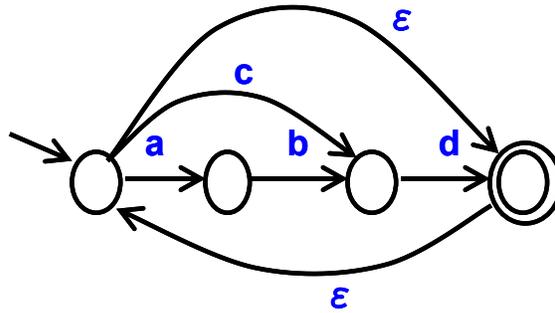
        1* 0 1*(01*01*)*

        Basically a single 0, combined w/ RE for even # of 0s, mixed with lots of 1*

    b. (4 pts) Give a NFA that accepts all binary numbers (strings of 0s and 1s) with an even number of 0s.
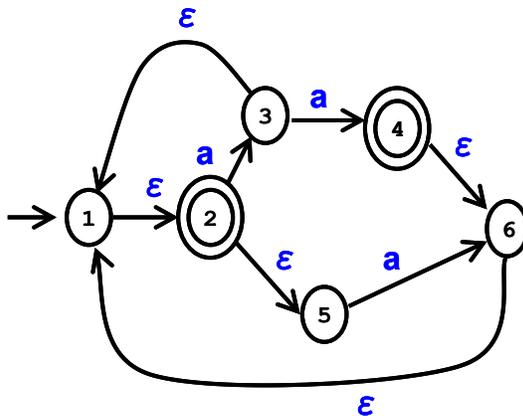
        Many possible answers, one example:

c. (6 pts) Create a NFA for the regular expression ((ab|c)d)* using any method.

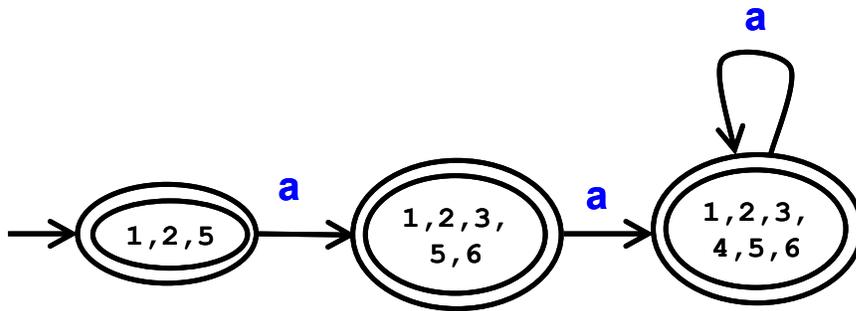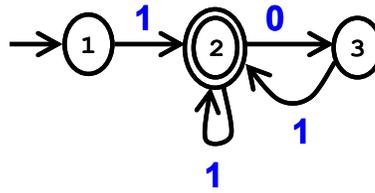**Many possible answers, one example:**

3. (18 pts) NFA to DFA

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

Answer:

(8 pts) DFA Minimization. Consider the following DFA.

.

a. (2 pts) What criterion is used to initially place DFA states in one of 2 partitions?

Whether state is a final state.

.

b. (6 pts) Assume states 1 & 3 are in one partition, and state 2 is in a different partition. Are states 1 & 3 distinct (i.e., should be placed in 2 different partitions)? Explain.

No, states 1 & 3 are not distinct. Both states behave in the same manner for all transitions: reject for input "0" and enter partition containing 2 for input "1",

4. (8 pts) Context free grammars

a. (4 pts) Provide a *rightmost* derivation of the string "110" for the following grammar: S → SS | 1 | 0

S ⇒ SS ⇒ S0 ⇒ SS0 ⇒ S10 ⇒ 110   OR   S ⇒ SS ⇒ SSS ⇒ SS0 ⇒ S10 ⇒ 110

b. (4 pts) Create a context free grammar for binary numbers (strings of 0s and 1s) with exactly one more 1 than 0s. Your grammar may be ambiguous.

Many possible answers, one example:
S → E1E
E → E1E0E | E0E1E | EE   # Same numbers of 0 & 1s

5. (6 pts) Ruby features

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

a. "330" =~ /(3+) /   **# Output = 33 nil**
    puts $1
    puts $2

b. a = { }   **# Output = b**
    a["b"] = 3
    a.keys.each{ |x| puts x }

6. (36 pts) Ruby programming

In this problem, you will write a short Ruby program that analyzes an assembly-like program stored in a text file. Each line in the file should have the form:

   <label>: <opcode> <operands...>

I.e., label followed by instruction (every line must have a label). There is one space to the right of the :, and one to the right of the opcode. Line numbers start at 1. The following table lists the three kinds of instructions and their operand formats

Instructions	ldc <reg>, <int>
	sub <reg>, <reg>
	bz <reg>, <label>

<label>	Sequence of lower case letters
<opcode>	ldc, sub, bz
<reg>	r0, r1, ..., r9
<int>	Sequence of digits

Implement a class *AssemblyProgram* with the following methods:

a. (22 pts). *compile*(filename) reads the contents of file "filename" and returns false if "filename" doesn't match the format above, or true if it does.
b. (7 pts) *find_label*(x) takes a label x and returns the line number of the statement with label x. If label x does not exist for any instruction, then *find_label* should return nil
c. (7 pts) *verify_labels* checks that all branch target labels exist (these are labels that are operands of bz instructions). It returns true if all labels are verified to exist, and false otherwise.

   For example, for the following program in file "prog"

```
                    # p = AssemblyProgram.new
start: ldc r0, 42   # p.compile("prog") = true, since prog has correct syntax
a: ldc r1, 37       # p.find_label ("a") = 2
b: sub r0, r1       # p.find_label ("start") = 1
c: sub r0, r2       # p.find_label ("branch") = 5
branch: bz r0, b    # p.verify_labels = true, since branch target "b" exists
```

You may assume that all instructions will have unique labels, and the functions *find_label* and *verify_labels* will be invoked only for programs for which *compile* returned true. The function *compile* may store information for use by *find_label* and *verify_labels*.

Helpful functions:	File.new(filename, mode)	// opens filename in mode, returns File
	file.eof?	// is File object file at end?
	file.readline	// read & return single line from file
	file.readlines	// read entire file, return array of lines

Many possible answers, one example:

```ruby
class AssemblyProgram

 def initialize
  @label = {}
  @targets = []
 end

 def compile(filename)
  lineNum = 1
  file = File.new(filename, "r")         # open file
  until file.eof? do                     # repeat until EOF
    line = file.readline                 # read lines from file
    if line    =~ /^([a-z]+): ldc r[0-9], [0-9]+$/
      @label[$1] = lineNum
    elsif line =~ /^([a-z]+): sub r[0-9], r[0-9]$/
      @label[$1] = lineNum
    elsif line =~ /^([a-z]+): bz r[0-9], ([a-z])+$/
      @label[$1] = lineNum
      @targets.push $2
    else
      puts "Syntax error #{line}"
      false
    end
    lineNum += 1
  end
  true
 end

 def find_label(name)
  @label[name]
 end

 def verify_labels
  @targets.each { |x| return false if @label[x] == nil }
  true
 end

end
```