CMSC330 Spring 2015 Midterm #1
9:30am/11:00am/12:30pm

Name:

_____

Discussion Time (circle one): 10am 11am 12pm 1pm 2pm 3pm
Discussion TA (circle one): Amelia Casey Chris Mike Elizabeth Eric Tommy

Instructions

- Do not start this test until you are told to do so!
- You have 75 minutes to take this midterm.
- This exam has a total of 100 points, so allocate 45 seconds for each point.
- This is a closed book exam. No notes or other aids are allowed.
- Answer essay questions concisely in 2-3 sentences. Longer answers are not needed.
- For partial credit, show all of your work and clearly indicate your answers.
- Write neatly. Credit cannot be given for illegible answers.

	Problem	Score
1	Regular Expressions	/17
2	Finite Automata	/14
3	NFA to DFA	/10
4	OCaml	/7
5	Programming Language Concepts	/12
6	Ruby execution	/20
7	Ruby programming	/20
	Total	/100

1. **Regular Expressions**. For parts b – e, below, assume the alphabet Σ = {a, b}.

   a. (3 points) Give an English description of the strings matched by Ruby regexp
   /[ct]h[a-z]*[aouei]$/

   A string ending with a substring starting with ch or th, followed by a lowercase letter, and then a lowercase vowel.

   b. (4 points) Give a Ruby regexp that denotes the language of strings in which the number of a's is divisible by three

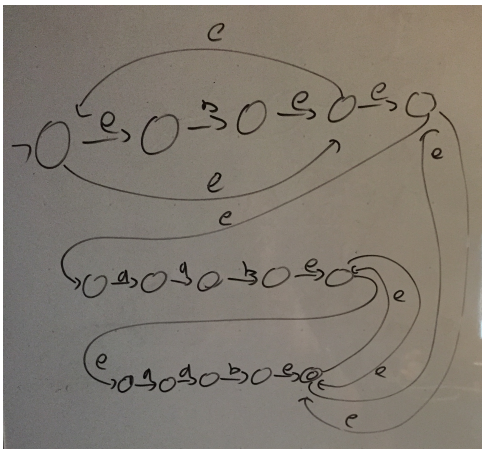
   b*(ab*ab*ab*)*

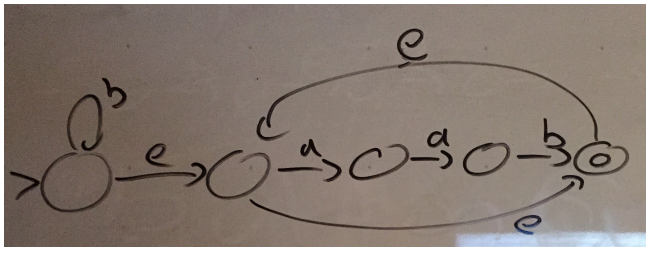
   c. (3 points) Give a Ruby regexp that denotes the language of strings that have exactly two bs.

   a*ba*ba*

   d. (7 points) Draw an NFA that recognizes the language defined by regexp b*(aab(aab)*)*

   The full one is the left, and a minimized one is on the right





2. **Finite Automata.** For the true or false questions, no explanation is needed; if you add one for consideration of partial credit, it could help or hurt you.

   a. (2 points) True or false: A DFA or NFA can detect if an input (of any length) is a palindrome. (A palindrome is a word that reads the same backwards as forwards.)

      False. (It can detect palindromes up to a fixed length, but not of any length.)

   b. (2 points) True or false: Epsilon transitions make NFAs more powerful than DFAs, in terms of the languages they can express.

      False. NFAs and DFAs can express the same languages (the regular ones)

   c. (2 points) True or false: Given an NFA with N states, the corresponding DFA always have 2^N states (assuming we explicitly include dead states).

      False. It is at most 2N states, not always. For example a DFA is also an NFA, for that NFA the corresponding DFA has the same number of states.

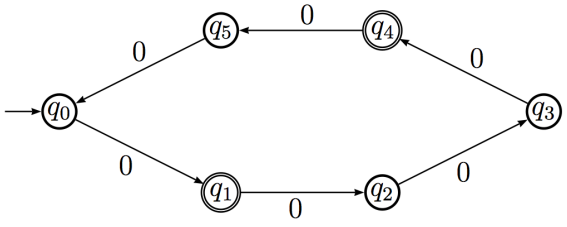
   d. Consider the following finite automata FA1 and FA2, depicted below, and answer the following questions:

      i. (3 points) Is either or both of FA1 and FA2 a DFA? Which one(s), if so?

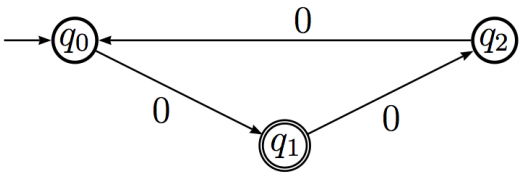
         Both.

      ii. (5 points) Do they recognize the same language? If not, give an example string that is accepted by one and not the other.

         Yes, they accept the same language.



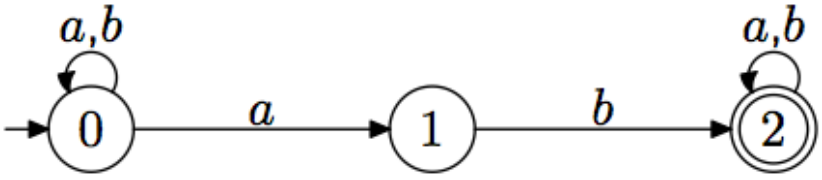
FA1
(final states are q1 and q4)



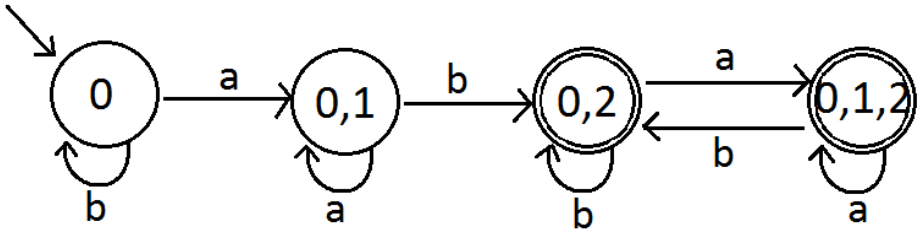
FA2
(final state is q1)

3. NFA to DFA conversion

a) (7 points) Convert the following NFA to a DFA using the subset construction algorithm. Be sure to label each state in the DFA with the corresponding state(s) in the NFA.



Answer:



b) (3 points) Give an English description of the strings matched by this NFA

Any number of a's and b's, followed by ab, followed by any number of a's and b's. I.e., all strings with ab as a substring.

4. OCaml

a. (3 points) Next to each of the following OCaml expression, give its type:

```
a) [2; 3]                    int list
b) [[1]; [2; 1]; [3]]        int list list
c) [(1, "a"); (2, "b")]      int*string list
```

b. (4 points) When executing the following code, what values are `y` and `z` initialized to?

```
let x=5 in
let y=x+1 in          y is 6
let x=2 in            (overrides previous x, does not assign to it)
let z=x+y             z is 8
in z;;
```

5. Programming Language Concepts

a. (3 points) Circle each programming language attribute/feature which describes **Ruby**:

Object-oriented Static typing Pattern matching **Implicit declarations** Overloading

Note: Regular expression matching is not pattern matching

b. (3 points) Circle each programming language attribute/feature which describes **OCaml**:

Object-oriented **Static typing** **Pattern matching** Implicit declarations Overloading

Note: Object-oriented was optional: circling it neither helped or hurt you

c. (3 points) Which is true of languages that employ dynamic typing? Circle all that apply:

I. Type errors will not be caught until you run the program

II. Type errors are not caught at all – too dynamic

III. At different times, variables can contain values with different types

IV. Types are inferred by the compiler, so they don't need to be written down

V. Lists are required to contain elements all having the same type

b. (3 points) True or false: Ruby's == method is equivalent Java's == method. Explain.

False. Ruby's == method checks structural equality, while Java's checks pointer/reference equality. Another difference is that == can be overridden in Ruby, but not in Java.

6. Ruby execution. Write the output, if any, of executing the following code. Write FAIL if an error will occur at run-time (following any output to that point).

a. (4 points)
```
x = [1,2, nil, 4]
x[3] = 3
x.each { |e| if e then print e else print "?" end }
```

12?3

b. (4 points)
```
class F
  def initialize(x)
    @@g = x
  end
  def foo(z)
    return @@g + z
  end
end
a = F.new(1)
b = F.new(2)
puts a.foo(5)
```

7

c. (4 points)
```
ss = ["abc", "def", "ghij", "???", "WXYZ" ]
ss.each { |s|
  if s =~ /^(a|d|W)..$/ then
    print $1
  else
    print "X"
  end
}
```

adXXX

d. (4 points)

```
arr = [2, 1, -3, 5]
arr.sort
arr2 = [6, 2, 1, 1]
arr2.sort!
puts arr
puts arr2
```

2 1 -3 5 1 1 2 6 (each on its own line)

e. (4 points)

```
h = { 1 => “hello”, 2 => “bye” }
x = h.keys.collect { |k| h[k] }
puts x[0]
```

hello or bye were acceptable, since there is no guarantee of the order the keys are provided.

7. Ruby programming. (20 points) A *sparse array* is one that in which most of the elements of the array consist of the same (default) value. If the array is very large, then representing it as we normally would require unnecessary space.

Implement a class `SparseArr`. It implements a two-dimensional, square array whose representation assumes it will be sparse. Complete the implementation of the array. Importantly, the space occupied by your array should be *O(n)* where *n* is the number of elements you have inserted into the array (note that there can be somewhat large constant factors here). You should also strive for *O(1)* lookups and updates, i.e., get and put calls should take roughly constant-time. ***Hint***: you can assume that Ruby hashes have these properties, so you might want to use them in your implementation (but you can use other data structures, too, if you like).

Methods to implement:

the constructor takes the size of the array (the dimension), and the default value

`size` returns the dimension of the array

`get(x,y)` returns the value at coordinate x,y, or throws an exception if out of bounds

`put(x,y,e)` updates the value at x,y, or throws an exception if out of bounds

`+(arr)` returns a new array that is the element-wise sum of the current array and arr; throws an exception if arr and the current array are not the same size; the new array's default element matches the current one's

Example session:

```
irb> x = SparseArr.new(2,0)
=> ...
irb> x.size
=> 2
irb> x.get(1,2)
IndexError: 1,2 outside of bounds of 2D-array size 2
  ...
irb> x.get(0,0)
=> 0
irb> x.put(1,1,5)
=> 5
irb> x.get(1,1)
=> 5
irb> (x + x).get(1,1)
=> 10
```

Note that you should maintain the sparseness of the array, in your implementation. For example, the array produced by x+x in the above interaction should also be sparse

```
class SparseArr
  attr_reader :size
  def initialize(size,elem)
    @arr = Hash.new
      (* you could also have set the hash's default value: Hash.new(elem) *)
    @elem = elem
    @size = size
  end
  def get(x,y)
    checkbounds(x,y,@size) (* defined on next page *)
    z = @arr[[x,y]]
    if z == nil then @elem else z end
  end
  def put(x,y,e)
    checkbounds(x,y,@size)
    if get(x,y) != e then (* preserves sparseness if e is default value *)
      @arr[[x,y]] = e
    end
  end
  def +(arr)
    if arr.size != @size then
      raise IndexError, "#{arr.size} of argument != array size #{@size}"
    end
    r = SparseArr.new(@size,@elem)
    for x in 0...@size do
      for y in 0...@size do
        v = get(x,y)
        w = arr.get(x,y)
        r.put(x,y,v+w)
      end
    end
    r
  end
end
```

```
# you might find this useful: raises an exception if either x or y is out of the bounds
  def checkbounds(x,y,sz)
    if x < 0 || y < 0 || x >= sz || y >= sz then
      raise IndexError, "#{x},#{y} outside of bounds of 2D-array size #{sz}"
    end
  end
```

Standard library methods:

x.push(e) adds e to the end of x, an array
x.unshift(e) adds e to the start of x, an array
x.shift returns and removes the element at the front of x, an array
x.pop returns and removes the element at the end of x, an array
x.each takes a code block expecting one argument. The method will invoke the code block for each element in x, an array, and will return the array, unchanged
x.collect takes a code block expecting one argument. The method will invoke the code block for each element in x, an array, and collect the results in an array that's returned
x.flatten is a one-dimensional "flattening" of x, an array, which merges the contents of any nested arrays into x, directly
h.each takes a code block expecting two arguments. The method will invoke the code block for each key-value pair in h, and leave h unchanged
h.collect takes a code block expecting two arguments. The method will invoke the code block for each key-value pair in h, and collect its results in a list, leaving h unchanged
h.empty? returns true if h, a hash, is empty, and false otherwise