CMSC 330 Exam #1 practice questions #1– solutions Fall 2012

1. [16 pts.]

   a. Here's an example illustraing that Ruby has dynamic typing:

   ```
   x= 330
   x= "CMSC 330 is lots of fun!!!"
   ```

   Note that dynamic typing isn't the same thing as implicit variable declarations. Ruby has implicit declarations, meaning that variables don't have to be declared, but there are languages that have static typing that have implicit declarations. To show that Ruby has dynamic typing it's necessary to show that types aren't checked until runtime, or an example of something's type being able to change during runtime.

   b. Here are several differences between NFAs and DFAs:

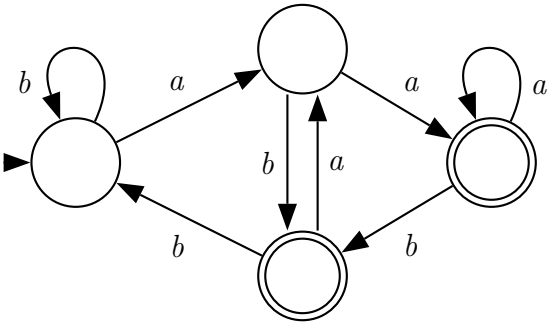
   - NFAs can have ϵ–transitions, but DFAs can't.
   - NFAs can have zero or more transitions from any state on any alphabet symbol, while DFAs must have exactly one transition from every state on every alphabet symbol.
   - An NFA can have zero or more paths from the start state to a final state on a string, while a DFA will always have exactly one path from the start state for any string.
   - An NFA accepts a string if there is some path from the start state to a final state on the string, even if other paths for that string don't lead to a final state, while a DFA accepts a string if the single path from the start state leads to a final state.

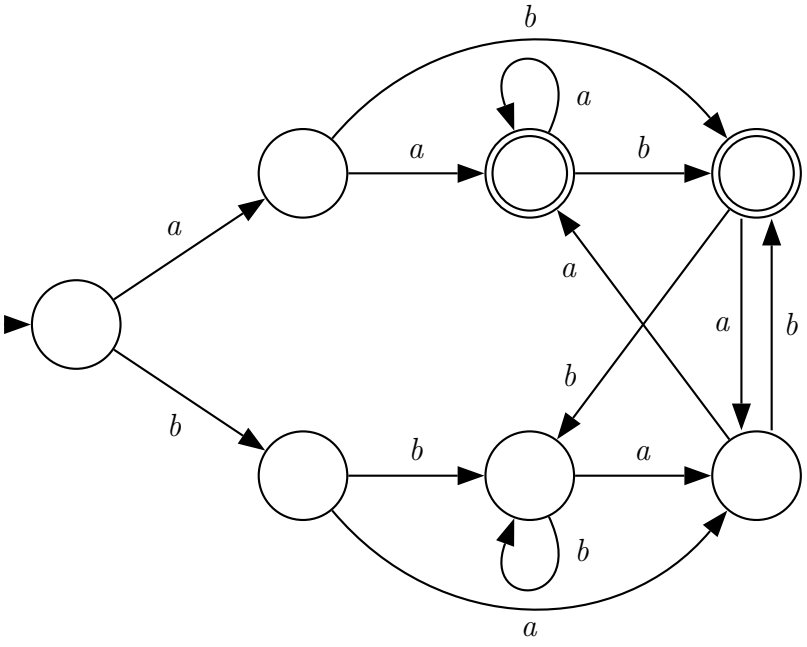
   c. To say that a function has *polymorphic* type means that it can operate upon parameters if different types.

   Note that if a function has polymorphic type it doesn't mean that its parameters can be **any** types, just that it can operate upon parameters of different (more than one) type. For example, in OCaml a function with type `'a list -> int` cannot be called with an `int`, a `string`, or a tuple as parameters– so clearly the argument can't be any type at all. It must be passed a list, but lists with any type of elements are acceptable.

   d. To say that a language has *higher–order functions* means that it allows functions to be passed as function parameters or returned as function return values (or bound to variables).

2. [24 pts.] Two correct DFAs are shown below; of course other correct versions exist as well.





3. [18 pts.] Different answers are possible; here is one:

```
count= 0

while (line= gets)
  if line =~ /^\(S0,[a-z],[A-Za-z0-9]+\)$/ then
    count += 1
  end
end

print(count, "\n")
```

It would also have been possible to use something like `([A-Za-z0-9]+)` in the regular expression where `S0` appears above, and use a backreference later to compare `$1` to the string `"S0"`. Using Ruby's `String.split` method to break apart the transition would not have been the easiest approach, because then the validity of each of the three parts of the transition would have had to be checked for validity separately.

4. [26 pts.] Different answers are possible; here are several:

$\Big(aa^*(b|c) \;\Big|\; b \;\Big|\; c \;\Big|\; dd^*(b|c) \Big)^* \, (a^* \mid d^*)$

$\Big(a^*(b|c) \;\Big|\; d^*(b|c) \Big)^* \, (a^* \mid d^*)$

$\Big(\big((a|b|c)^* \, (b|c) \, (b|c|d)^* \big)^* \;\Big|\; a^* \;\Big|\; d^* \Big)$

$\Big((a^*|d^*) \, (b|c) \Big)^* (a^* \mid d^*)$

5. [16 pts.]

a.
```
let rec nth (n, l) =
      if n = 1
        then match l with
          (h::t) -> h
        else match l with
          (h::t) -> nth ((n - 1), t)
```

b. `int * 'a list -> 'a`