**Tutorial 1: Regular Expressions and FSA**

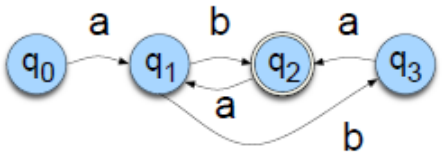
Q1. Write regular expressions for the following languages. By "word", we mean an alphabetic string separated from other words by whitespace, any relevant punctuation, line breaks, and so forth.

1. The set of all alphabetic strings;
2. The set of all lower case alphabetic strings ending with a letter *b*;
3. The set of all strings with two consecutive repeated words (e.g., "Humbert Humbert" and "the the" but not "the bug" or "the big bug");
4. All strings that start at the beginning of the line with an integer and that end at the end of the line with a word;
5. All strings that have both the word *grotto* and the word *raven* in them (but not, e.g., words like *grottos* that merely *contain* the word *grotto*);

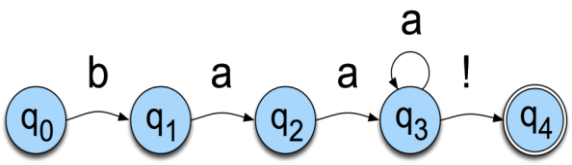
HINT: please consult the book Chapter 2.1 or some websites on regular expressions.

Q2. Design an FSA that accepts a subset of valid web addresses. Be sure to accept addresses from the ".com" and ".org" domains, and handle an arbitrary amount of directory nesting. Accept at least ".html", ".htm" and ".shtml" page types. (Interested students can consult the official Web standards for other possible extensions to this recognizer.)

Q3. Write a regular expression for the language accepted by the following NFSA.



Q4. Given the FSA in the following figure, walk through the D-RECOGNIZE algorithm on strings baaaa*!* and *baac*, respectively.




Q5. Currently the function D-RECOGNIZE (as in Fig. 2.12 in the textbook) solves only a subpart of the important problem of finding a string in some text. Extend the algorithm to solve the following two deficiencies:

a. D-RECOGNIZE assumes that it is already pointing at the string to be checked, and

b. D-RECOGNIZE fails if the string it is pointing to includes as a proper substring of a legal string for the FSA. That is, D-RECOGNIZE fails if there is an extra character at the end of the string.

c.

```
function D-RECOGNIZE(tape, machine) returns accept or reject

  index ← Beginning of tape
  current-state ← Initial state of machine
  loop
    if End of input has been reached then
      if current-state is an accept state then
        return accept
      else
        return reject
    elsif transition-table[current-state,tape[index]] is empty then
      return reject
    else
      current-state ← transition-table[current-state,tape[index]]
      index ← index + 1
  end
```