# Homework 2

Out: 29 Jan. Due: 5 Feb.

**Instructions:** *Submit your solutions in pdf format on Gradescope by* **5pm on Friday, February 5**. *Solutions may be written either in LaTeX (with either machine-drawn or hand-drawn diagrams) or* **legibly** *by hand. (The LaTeX source for this homework is provided in case you want to use it as a template.) Please be sure to begin the solution for each problem on a new page, and to tag each of your solutions to the correct problem! Per course policy, no late solutions will be accepted. Take time to write* **clear** *and* **concise** *answers; confused and long-winded solutions may be penalized. You are encouraged to form small groups (two to four people) to work through the homework, but you* **must** *write up all your solutions on your own. Depending on grading resources, we reserve the right to grade a random subset of the problems and check off the rest; so you are advised to attempt all the problems.*

**1.** An *all-paths* NFA is defined in the same way as a standard NFA, except the definition of acceptance is changed so that the machine accepts a string $w$ if and only if *all* possible computations on input $w$ lead to accepting states. (In other words, if *any* computation on $w$ dies out or ends in a rejecting state, the machine rejects; else it accepts.) Show that the class of languages accepted by an all-paths NFA is exactly the class of regular languages. [HINT: One direction follows immediately. For the other direction, to convert an all-paths NFA into an equivalent standard NFA, think about the "subset" construction we used to convert an NFA into a DFA.]

**2.** Let $L$ be a regular language over some alphabet $\Sigma$. Show that the language

$$\tfrac{1}{2}(L) = \{x : \exists y \text{ s.t. } xy \in L \text{ and } |x| = |y|\}$$

is also regular. [HINT: This problem is quite challenging and you may want to leave it until last. Given a DFA for $L$, construct an NFA for $\frac{1}{2}(L)$. The "product construction" used in the proof in class that regular languages are closed under intersection should be useful here.]

**3.** Construct regular expressions that denote each of the following languages:

(a) The set of all words over the English alphabet that begin with a vowel (i.e., with 'a', 'e', 'i', 'o' or 'u') and end in 'ing'. [NOTE: Use the symbol $\Sigma$ as shorthand for the regular expression $\mathbf{a} \cup \mathbf{b} \cup \ldots \cup \mathbf{z}$, denoting the English alphabet. Don't forget that the string 'ing' itself is in this language!]

(b) The set of all 0-1 strings in which the number of 1's is divisible by three.

(c) The set of all 0-1 strings such that in any prefix, there is at most one more 1 than 0's and at most one more 0 than 1's. [HINT: Start by writing down a regular expression only for the *even-length* strings in this language. Then extend it to all strings.]

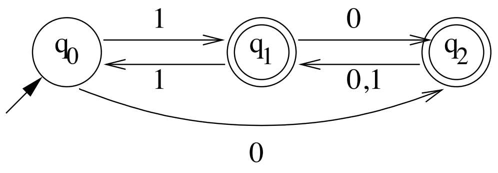
**[continued on next page]**

**4.** Let $R, S$ be arbitrary regular expressions. Which of the following statements are true? If the statement is true, give a proof; if it is false, give a simple counterexample.

(a) $(R \cup S)^* \equiv R^* \cup S^*$; (b) $(R^*)^* \equiv R^*$; (c) if $R^* \equiv S^*$ then $R \equiv S$.

NOTE: The symbol '$\equiv$' denotes *equivalence* of regular expressions, i.e., $R \equiv S$ means that $R$ and $S$ denote the same language, or $L(R) = L(S)$.

**5.** Construct a regular expression that is equivalent to the following DFA, using the procedure discussed in class. Identify clearly each stage in your construction. Remove the states in the order $q_0, q_1, q_2$.



Optional: Can you find a simpler regular expression that denotes the same language? (Don't spend a lot of time trying to find a much simpler expression!)

**6.** *Note: For this problem, you will need access to a UNIX or LINUX shell. You will probably already have this from (e.g.) CS61B; if not, then you can find an online terminal (no need to install!) by searching on Google.*

Regular expressions are used in several common utilities, such as `grep` and `lex` in UNIX and in the search functions of various text editors. This exercise illustrates this point with reference to the UNIX file-searching utility `egrep`. You should start by carefully reading the manual page for `egrep` (using the command "`man egrep`"). Notice that the syntax for regular expressions used by `egrep` contains some variations and extensions of that used in class. You will also need to download the plain text data file from the class web page.

For each of the following conditions, use `egrep` to print out all lines in the data file that satisfy the condition. Submit a listing of all five outputs, showing clearly the `egrep` command you used to generate them.

(a) Lines containing at least one decimal digit.

(b) Lines containing both of the strings "dome" and "pleasure".

(c) Lines containing at least two occurrences of the pattern "and ". (Note the space character at the end of this pattern.)

(d) Lines containing at least one comma, period or exclamation point. [NOTE: Since some of these punctuation marks are part of the `egrep` regular expression syntax, you should precede them with the escape character backslash.]

(e) Lines *not* containing any commas, periods or exclamation points. [HINT: You could do this by writing the complement of the regular expression you used for part (d), but that is a bit messy. For an easier solution look at the options on the `egrep` man page.]