CS 164, Fall 2006 **CS 164: Homework #1** **P. N. Hilfinger**

Due: Friday, 30 January 2009, 2400

General instructions about homework. Please submit this homework electronically. See the on the class web page. You will need Subversion and you will need to have electronically registered and to have generated an SSH key (logging into your class account does this). In any case, please notify us of problems.

You'll find templates for some solutions in the directory `~cs164/hw/hw1` and in the repository at

`svn+ssh://cs164-ta@quasar.cs.berkeley.edu/staff/hw1`

Place answers to any questions that don't require programs in a file called **hw1.txt**.

If you are working at home using your own installation (as opposed to using the instructional machines remotely), you'll need a Python installation. See the Python link from the class home page if you need one. Problems 4 and 5 call for the use of the **fsasim** program, which is a Python program that is available from your instructional account (you can download it from `~cs164/bin`; at home, you'll have to change the first line to wherever you have Python installed, or run fsasim using the 'python' command). There is a manual for **fsasim** available from the CS164 home page.

1. A *subsequence* of a string S (or any kind of sequence, really) is simply a sequence consisting of zero or more characters from S in the same order as in S; for example, `"ack"`, `""`, `"bk"`, and `"back"` are all subsequences of `"back"`, but `"kb"` and `"bb"` are not. A *substring* of S is a *contiguous* subsequence of zero or more characters of S; for example `"ack"`, `"ba"`, `""` and `"back"` are substrings of `"back"`, but `"bk"` is not. Given a string S of length N, how many subsequences does it have? How many substrings? To simplify your life, you *may* assume that letters in S are not repeated (the problem becomes rather "interesting" if you count only distinct subsequences when S may contain repetitions).

2. [From Aho, Sethi, Ullman] Give as simple a description as possible of the languages denoted by the following regular expressions. For example, it is better to describe the language denoted by `((a*b*)*(b*a*)*)*` as "The set of all strings from the alphabet $\{a, b\}$" rather than "A sequence of 0 or more a's followed by 0 or more b's, all repeated 0 or more times and then followed by *etc.*" The latter might be correct, but shows no thought.

a. `0(0|1)*0`

b. `((ϵ|0)1*)*`

c. `(0|1)*0(0|1)(0|1)`

d. `0*10*10*10*`

e. `(00|11)*((01|10)(00|11)*(01|10)(00|11)*)*`

More problems on next page.

3. [Adapted from Aho, Sethi, Ullman] Write the simplest regular expression you can for each of the following languages, using basic regular expressions from Python. Turn in files called **P3a.py**, **P3b.py**, etc., each containing that contain solutions to these problems, using the templates for these files provided in **~cs164/hw/hw1** on the instructional machines. Python "regular expressions" are considerably more powerful than the "classical" basic expressions we want here. For this problem, restrict yourself to using just ordinary characters (that stand for themselves), plus the special characters

```
    [ ] ( ) * + | . $ ^ ?
```

In the following, "letters" mean lower-case letters.

a. Strings of letters that contain all five vowels in order, exactly once.

b. Strings composed of letters a–f in which the letters are in alphabetical order (not all letters have to appear in a string, but those that do must be in order).

c. Comments consisting of a string starting with `/*` and ending with `*/` without any `*/` in between unless it occurs inside balancing quotes `"` and `"`. Quotes must be balanced, so that `/*"*/` is illegal, and only balancing pairs of quotes "deactivate" the `*/` symbol, so that `/*""*/""*/` is illegal (the `*/` occurs between quotes, but they don't balance each other).

d. All strings of 0's and 1's with an even number of 0's and an odd number of 1's.

e. All strings of 0's and 1's that do not contain the substring '011'.

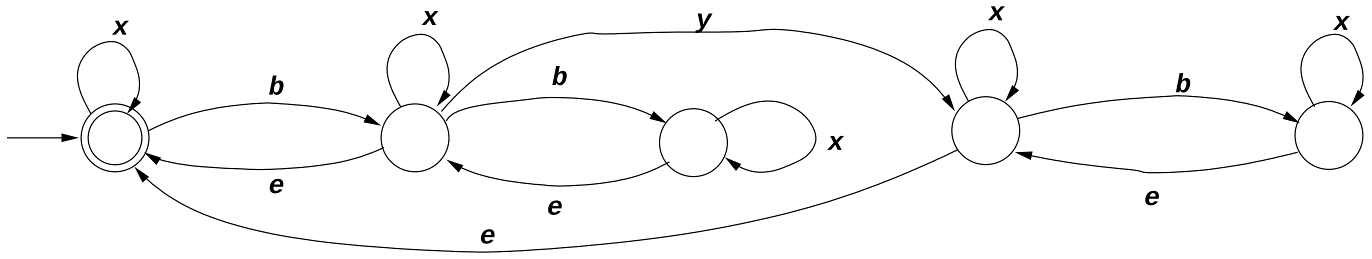
f. All strings of 0's and 1's that do not contain the subsequence '011'.

4. Find the simplest NFA you can for problem 2c, above. Turn in a file called **2c.nfa** containing your answer in **fsasim** format.

5. Find the simplest DFA you can for 2c. Turn in a file called **2c.dfa** containing your solution in **fsasim** form (see above).

6. Suppose that I have two NFAs, M_1 and M_2, recognizing the languages L_1 and L_2, respectively. Give a general algorithm for constructing an NFA that recognizes the language $L_1 - L_2$, which is the set of all strings in L_1 that are not in L_2. You don't need to write a program, just the give the algorithm in sufficient detail that a reasonable programmer could fill in the details.

7. Give the simplest description you can of the language described by the DFA below:



8. The authorization file we use for Subversion, which tells which users may access which directories, contains entries that look like this:

```
[/DIR]
login1 = rw
login2 = r
login3 = rw
* =
```

where *logini* are ordinary Unix login names. (This entry means that *login1* and *login3* have read-write (rw) access to the directory *REPOSITORY/DIR*, *login2* has read-only (r) access, and everyone else (denoted '`*`') has no access. As shown, these are all on separate lines, and otherwise whitespace may be used freely. The end of the list of authorized users for a given directory is marked by the next directory (`[...]`) line, or the end of file. Write a program using full Python regular expressions to search such a file for cases where there are (at least) two entries with the same *DIR* and remove all but the last entry, using the template file in cs164/hw/hw1/cleanup.py. You will probably want to consult the entry in the Python library reference documentation for the `re` module.