HW 3: Regular Expressions and Finite Automata

CS 440: Programming Languages and Translators
Due Sat Feb 16, 11:59 pm

Small Change to Turn-in Procedure
This change is to make the TA’s lives a bit easier: Instead of having one person in a group turn in the assignment, please have each person in the group turn in the assignment. Each person should list all the people in the group.

Problems [50 points total]

1. [8 = 2 * 4 points] Give simple English descriptions for the languages of the regular expressions below. \
   \[ \d \text{ stands for } [0-9] \]
   1a. \((19|20)\d\d- (0? [1-9]| [12]\d | 3 \{01\}) \]
   1b. \#17: \[ 1-9g \] \[ 0-9h \]* (e [+-]? \[1-9]\d*) \)
   \#17: \[ 1-9a-h \] \[ 0-9a-h \]* (e [+-]? \[1-9]\d*) (Updated Thu 2/14)

   (Thu 2/14) The dashes in \[ 1-9 \], \[ 0-9a-h \] etc. stand for sequences: \[ 1-9 \] is short for \[ 123456789 \], \[ 1-9a-h \] is short for \[ 123456789abcdefh \] (and \[ 0-9a-h \] is similar but adds 0). The dashes in \d\d- and (e [+-] ...) are actual dashes. (For [+-], the dash doesn’t indicate a sequence because there’s nothing to its right. Similarly, [-+] wouldn’t stand for a sequence either. (It’s equivalent to [+-].) The # and : mean literally # and :.

2. [8 = 2 * 4 points] Write some fairly complicated expressions on strings. Give regular expressions for each of the following sets of strings. Wed 2/13
   2a. Strings \( \in \{a, b\}^* \) where the number of a’s is even or the number of b’s is divisible by 3. [Hint: write it as (expr for desired number of a’s) | (expr for desired number of b’s)

   Random examples: b, aa, baabbb, aba, a, bbb, aabbaaba, abbbababbaa
   2b. Strings \( \in \{a, b, c\}^* \) that don’t contain the 3-character substring abc.

3. [12 = 3 * 4 points] Give a regular expression for numerals in the following formats. You can use \d.
   3a. Natural numbers with no leading zeros (except for just 0)
   3b. Integer with optional sign and optional exponent (e with optional + or – and natnum), where natnum is the expression from part a.
   3c. Floating point numbers: Include numbers like 1.2, 1., .2 and an optional exponent.
   3d. Hexadecimal constants prefixed with 0x. Leading zeros, a–f, and A–F are allowed.

4. [10 points] Use the algorithm in section 2.8.2 of Algorithms for Compiler Design to convert the regular expression \(( (b | c) | a (a | c) )^* \) into a nondeterministic finite automaton. (You can drop redundant
ε transitions, if you like, but you're not required to.) By the way, the weird φ expression at the top of the table in 2.8.2 is not the same as the expression ε; expression φ is for the empty regular expression (which doesn’t come up very often).
5. [12 points] Convert the NFA below to an equivalent DFA.
   First take the $\epsilon$-closure of the NFA to get one without $\epsilon$-transitions (the discussion after Figure 2.6 is useful here). Then get rid of any stuck paths by adding an error state and transitions to it. Then use the DFA-state as set of NFA-states technique to complete the conversion (the discussion before Figure 2.3 is useful here).