CS385-Assignment 4
Due Thursday, October 4

This includes both problems on deriving regular expressions from automata, and some problems on proving languages to be nonregular (including the pumping lemma).

1 Exercises and Problems from the Text
1.21, 1.22, 1.29(b), 1.46(c), 1.48,1.53.
Hints and Comments. The standard intuition suggests that the language of 1.48 is not regular, since it would appear that the state of the machine has to store an arbitrarily large count. The result you are asked to prove says that this intuition is WRONG, so that there must be other ways to recognize this language. Write down a few strings that are in the language of 1.48, and a few that are not, and you should soon see that there is another way to characterize this language, which makes its regularity apparent. Remember that to show that a language is regular you merely have to exhibit an automaton that recognizes it.

Study the worked solutions for 1.29(a,c) and 1.46(b) carefully. For problems like 1.46 and 1.53, there are several strategies: You can try to apply the pumping lemma directly, but this often requires a very careful choice of the string to be pumped. Often it is easier to use closure properties to argue “if \( L \) is regular, then some other language \( L' \) is regular”, and apply the pumping lemma to the simpler \( L' \). This latter approach works very well for the assigned problems. For example, in 1.53, think about the set of the strings in \( ADD \) in which one of the summands is 0 and the other summand consists entirely of 1’s. This is the intersection of \( ADD \) with some regular language. (Which one?)

2 Problems with grep
These are exercises in the use of the UNIX filter grep. If your linux account is still active (are linux accounts still active?) you can use that, otherwise, use the Terminal utility on one of the Macs in the lab.

\texttt{grep -E 'regular expression$' filename}

the program will print all the lines of the named file that match the regular expression. There are special rules for forming regular expressions in grep:
1. A period matches any single character.
2. You can specify a set of characters by writing the characters in brackets. Thus the expression \( [abc] \) matches any one of the three characters a,b,c. (In our usual notation for regular expressions, we would write \( a \cup b \cup c \).) You can also specify a range of characters: \( [a-z] \) matches any lower-case letter.
3. You can specify an excluded set of characters as follows: \( ^{[aeiou]} \) matches any single character other than the five lower-case letters in the brackets. Note that the symbol \(^\) is NOT a general complementation operator—we aren’t allowed complementation or intersection in regular expressions.
4. The vertical line \( | \) denotes union. For example, \( (a|b|c) \) is equivalent to \( [abc] \). Concatenation is indicated by writing the two expressions adjacent to one another, and star by writing an asterisk immediately after the expression. If \( r \) is an expression, you can write \( r^+ \) as an abbreviation for \( rr^* \). You can, and should, use parentheses.

**Examples**

```
  grep -E '^[^aeiou]*a[^aeiou]*a[^aeiou]*$' /usr/share/dict/words
```

finds all words whose only vowel is a, and in which a occurs exactly three times. These include cataract and sassafras.

Here are the problems for you to do. Use grep to find all words in \(/usr/sharedict/words\) that

(a) Contain at least two non-consecutive z’s (for example zigzag).
(b) Contain at least two g’s and at least one m (for example, muggy).
(c) Contain at least three i’s and no other vowels (for example, diminish).
(d) Contain none of the vowels a,e,i,o,u.
(e) Contain at least six letters that are either s or t.

Hand in printouts of both the commands you typed in grep to get the answers to these questions, and the lists of words you found. If you get a very long list for any of these queries, you only have to hand in the first twenty or so words found.