CS 383
Exam 2 Solutions
April 20, 2007

1. Here are some languages:
   a) Strings of 0's and 1's where every 0 is followed immediately by a 1.
   b) $\{0^n | n \geq 0\}$
   c) $\{0^n 1^n | n \geq 0\}$
   d) $\{0^n 1^n 2^n | n \geq 0\}$
   e) $\{0^n 1^n 2^n 3^n | n \geq 0\}$
   f) Strings whose lengths are multiples of 3.
   g) Strings of odd length whose center element is 0.

Mark with an R the languages that are Regular, with a C the languages that are Context Free, and with a G (for General) the languages that can be accepted by a Turing Machine.

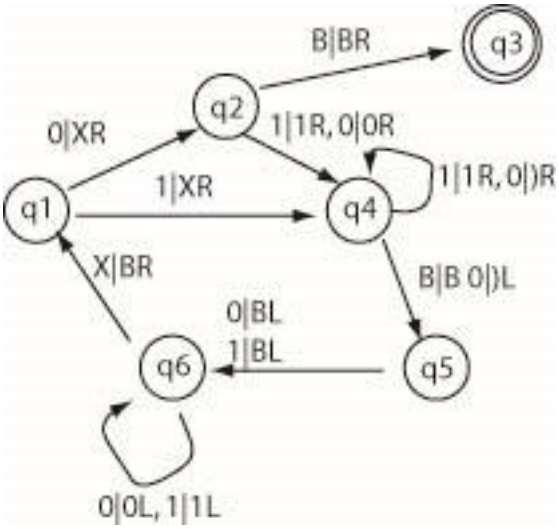
a) **RCG**
b) **RCG**
c) **CG**
d) **G**
e) **G**
f) **RCG**
g) **CG**

2. Let $\mathcal{L}$ be the language of strings of 0's and 1's such that at least a third of the digits in each string in $\mathcal{L}$ are 0 and at least a third of the digits are 1. For example, 001111 and 1010101 are both strings in $\mathcal{L}$. Show that $\mathcal{L}$ is not context free.

   Sadly, this is a mistake. The proof I expected you to use starts with $z = 0^p 1^{2p}$ and this string actually is pumpable: just use v = 0, x = 11. I gave everyone credit for this problem.

3. Design a Turing Machine that accepts strings of 0's and 1's where the length of the string is odd and the center element of the string is 0. For example 00000, 11011, and 10011 are all strings in this language. You can use any kind of TM that suits you: multiple tracks, multiple tapes, etc. (My solution uses a standard TM so you don't have to use a fancy machine, but you are welcome to do anything that works.)

   There are many ways to do this. I handle 0's and 1's at the start separately. On 0's I look to see if the next character is a blank; if so I am at the middle of the string and I accept. On any other character, and on a 1 at the beginning, I overwrite with an X, then go to the end of the string and write blank over the last character. I go back to the beginning and erase the X, and start over:



4. Here is a grammar $\mathcal{G}_1$:

   S -> 0 S 0 | 1 S 1 | A

   A -> 0 | 1 | ε

   Here is another grammar $\mathcal{G}_2$:

   S -> Z S_1 | Y S_2 | ZZ | YY | 0 | 1

   S_1 -> S Z

   S_2 -> S Y

   Z -> 0

   Y -> 1

   a) What does it mean when we say that $\mathcal{G}_2$ is the Chomsky Normal Form for $\mathcal{G}_1$?

   It means that $\mathcal{G}_2$ accepts the same language as $\mathcal{G}_1$, except for the empty string, and that all the rules of $\mathcal{G}_2$ have the form A->BC, or D->d.

   b) Do $\mathcal{G}_1$ and $\mathcal{G}_2$ derive the same language?

   No. $\mathcal{L}(\mathcal{G}_2) = \mathcal{L}(\mathcal{G}_1)$-{e}

   c) Why do we care about Chomsky Normal Form?

   Parse trees for a grammar in Chomsky Normal Form are binary trees, which means that we can predict their height from their number of leaves. This is the key to proving the Pumping Lemma for Context Free Languages: if a string is long enough there must be a root-to-leaf path with a repeated variable.

5. You have probably noticed that we didn't have a pumping lemma for languages accepted by Turing Machines (i.e., Recursively Enumerable languages). Suppose $\mathcal{L} = \{0^{n^2} | n \geq 0\}$, which is certainly RE. Let w be a very long string in $\mathcal{L}$. This string contains only 0's, so it doesn't matter how many portions it is decomposed into. Could there be a portion of w that could be pumped any number of times, always producing another string in $\mathcal{L}$? Why or why not?

   Suppose we start with the string $w = 0^{n^2}$ for some large n and we pump a portion with a symbols. This means that the language will have strings of length n^2+ak for every k. It is easy to see that not all of these can be squares: the distance between successive squares gets larger and larger, while the distance between successive values of n^2+ak is always a. So this language $\mathcal{L}$ is RE, but not pumpable.