This homework is due by noon on Saturday, May 1, and will not be accepted late. It covers general grammars and Turing machines, Sections 4.6 and 5.1. Your answers to questions 1 to 4 can be turned in as a scanned PDF. The remaining questions ask you to construct Turing machines using an on-line web app. Your answers can be turned in in a .txt file; further instructions are given below.

1. (*4 points*) The grammar shown here on the left is for the language $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$. The grammar on the right is for the language $M = \{ww \mid w \in \{a,b\}^*\}$.

$$\begin{array}{rcl} S & \longrightarrow & XTZ \\ T & \longrightarrow & AbCT \\ T & \longrightarrow & \varepsilon \\ bA & \longrightarrow & Ab \\ CA & \longrightarrow & AC \\ Cb & \longrightarrow & bC \\ XA & \longrightarrow & aX \\ CZ & \longrightarrow & Zc \\ X & \longrightarrow & \varepsilon \\ Z & \longrightarrow & \varepsilon \end{array} \qquad \begin{array}{rcl} S & \longrightarrow & HTE \\ T & \longrightarrow & aAT \\ T & \longrightarrow & bBT \\ Aa & \longrightarrow & aA \\ Ab & \longrightarrow & bA \\ Ba & \longrightarrow & aB \\ Bb & \longrightarrow & bB \\ AE & \longrightarrow & Ea \\ BE & \longrightarrow & Eb \\ Ha & \longrightarrow & aH \\ Hb & \longrightarrow & bH \\ HE & \longrightarrow & \varepsilon \\ T & \longrightarrow & \varepsilon \end{array}$$

a) Using the grammar on the left, give a derivation for the string $aabbcc$, which is in L.

b) Using the grammar on the right, give a derivation for the string $abaaba$, which is in M.

2. (*3 points*) Create a general grammar for the language $\{a^n b a^n b a^n \mid n \in \mathbb{N}\}$, and indicate how the grammar works. You can show how the grammar works by giving comments on the rules. (Note that this language is similar to $\{a^n b^n c^n \mid n \in \mathbb{N}\}$. It is OK to have a grammar in which derivations can get "stuck." My grammar has 10 rules.)

3. (*4 points*) Create a general grammar for the language $\{www \mid w \in \{a,b\}^*\}$, and indicate how the grammar works. You can show how the grammar works by giving comments on the rules. (Note that this language is similar to $\{ww \mid w \in \{a,b\}^*\}$. Idea: Using rules similar to the above grammar for $\{ww \mid w \in \{a,b\}^*\}$, make strings like $abaabCDCCDHEabaab$, then instead of disappearing, the HE changes to a symbol that can convert the C's and D's to a's and b's. It is OK to have a grammar in which derivations can get "stuck," but it's not too hard to extend this idea to one that can't get stuck. My grammar has 28 rules.)

4. (*6 points*) Consider the language $L = \{a^{2^n} \mid n \in \mathbb{N}\}$.

a) Create a general grammar for the language $\{a^{2^n} \mid n \in \mathbb{N}\}$. The grammer contains all strings of a's whose length is a power of 2. (As a hint, note that if you start with one a and double it n times, then there will be 2^n a's. For full credit, write a grammar for which derivations cannot get "stuck." This can be done with a grammar that has seven production rules.)

b) Explain in words why your grammar works. How can it generate every string in L? Why can't it generate any other strings?

c) Using your grammar, write derivations for the strings a and $aaaaaaaa$. Note that $a = a^{2^0}$ and $aaaaaaaa = q^{2^3}$, so both of these strings are in L.

For the remaining exercises, you will turn in Turing machine programs that will work in the web app at http://math.hws.edu/eck/js/turing-machine/TM.html. You should turn in a plain text (.txt) file containing the programs. For some basic instructions on how to use the web app and how to save your work into a text file, see http://math.hws.edu/eck/js/turing-machine/TM-info.html. The program will be demonstrated in class on Wednesday, April 21. For a couple Turing machine programs, see the file http://math.hws.edu/eck/cs229/s21/TM-examples.txt, which has some examples that we will look at in class. There are several more examples that are already built into the web app.

---

5. (*3 points*) This is a small exercise to help you get used to working with the Turing machine simulator. In class, we looked at a simple example of a Turing machine that moves to the right searching for two $'s in a row. When (and if) it encounters them, it halts, and the machine is left on the second $.

Create such a Turing machine in the simulator. You should assume that the tape contains only a's, b's, blanks, and $'s.

This can be done using two states (in addition to the halt state), if you use the "stay" option (S) as a direction of motion at the end. Without that option, it requires 3 states to put the machine in the proper position at the end. If you use the "other" and "same" options for "Old Symbol" and "New Symbol" in some of your rules, you can do this with just four or five lines in the rule table.

6. (*4 points*) Create a Turing machine that checks whether the number of a's in a string of a's and b's is a multiple of 3. The input is a string of a's and b's with the machine positioned on the right end of the string. The output of the computation should be 1 if the number of a's is a multiple of 3, and should be 0 if the number is not a multiple of 3. Note that the b's don't contribute anything to the answer, but they need to be erased just like the a's need to be erased. (That is, the **only** thing left on the tape should be a 0 or 1, and the machine should be positioned on that 0 or 1.)

7. (*6 points*) In class, we looked at a "binary-to-unary" converter. The input is a string of 0's and 1's, considered to be a binary number. When started on the rightmost digit of a binary number, the output of the machine is a string of a's, where the number of a's is equal to the original binary number.

Write a "unary-to-binary" converter: When the machine is started on the right end of a string of a's as input, the output shoud be a binary number equal to the original number of a's. That is, the binary number should be the only thing on the tape, and the machine should be positioned on the rightmost digit of the binary number. Your machine does not have to work for empty input, but if you want to output the correct answer, 0, for empty input you can do it.

As an idea for the program, create the binary number to the left of the string of a's Erase an a from the right end of the string, move to the left end and increment the binary number, then move back to the right end of the string of a's.