CS 481 Homework #10

Question 1

Language accepted is $0(0 + 1)^*$
Problem 2. We build a deterministic Turing machine accepting the set \( \{ a^k | k \text{ is a power of 3} \} \) as follows: First, verify that our input is of the form \( a^* \); otherwise reject. Then check if our input is a single \( a \): if so, accept. Otherwise, mark the first \( a \) with a hat (\( \hat{a} \)), then go to the rightmost \( a \) and replace it with the symbol \( $ \). Our tape should now look like this:

\[
\vdash \hat{a}aaa...aaa$a$  \ldots
\]

Note that the hatted symbol is at position \( 3^0 = 1 \). Now we will enter a loop, after each iteration of which, the position of the hatted symbol will be the next power of 3, and all previous symbols will be ticked (\( \acute{a} \)). We proceed as follows:

1. Overwrite all the \( \acute{a} \) symbols with the symbol \( \bar{a} \).
2. While there are \( \acute{a} \) symbols remaining, overwrite the leftmost one with a \( \acute{a} \). Then find the leftmost unmarked symbol, tick it, and tick the symbol after it. If either is the \( $ \), then reject (its position is between two powers of 3).
3. When there are no \( \acute{a} \) symbols remaining, then the positions of the next two unmarked symbols are one less than a power of 3 and a power of 3, respectively. Find the \( \hat{a} \) symbol and overwrite it with a \( \acute{a} \). Find the leftmost unmarked symbol and tick it. Then hat the succeeding symbol. If we ticked the \( $ \), then reject (its position is one less than a power of 3). If we hatted the \( $ \), then accept (its position is a power of 3).

We repeat this loop until the machine accepts or rejects within it. Each time we iterate, each of the marked symbols causes two unmarked symbols to become marked. This is the mechanism for computing successive powers of 3.
3a. The following is a grammar that works for generating \( \{a^n b^n c^n \mid n \geq 1\} \).

\[
\begin{align*}
A & \rightarrow aABC \mid aBC \\
CB & \rightarrow BC \\
aB & \rightarrow ab \\
bB & \rightarrow bb \\
C & \rightarrow c
\end{align*}
\]

With \( A \) being the start symbol. Here we first generate \( a^n b^n c^k \), and then utilize the context sensitive-ness to move things around and convert them to terminals.

3b. The following grammar generates \( \{a^n^2 \mid n \geq 1\} \).

\[
\begin{align*}
S & \rightarrow DS \mid X \mid a \\
DX & \rightarrow AAXa \\
aX & \rightarrow Xa \\
DA & \rightarrow aAD \\
aA & \rightarrow Aa \\
AX & \rightarrow AA \\
Aa & \rightarrow aa
\end{align*}
\]

We are utilizing the fact that \( n^2 = 1 + 3 + \ldots + (2n - 1) \). The idea is that we first generate \( D^k X \). Each \( D \) then acts as an operator. On meeting \( X \), it generates two \( A \) and an \( a \). Other \( D \)'s act on suffixes of the form \( A^{2n-2} X a^{(n-1)^2} \) and generates \( 2n A \)'s and \( 2n - 1 a \)'s.

4. We first generate the LBA \( M \) from the CSG \( G \). That is given a string \( \vdash x \vdash \) on the input tape, we have to check whether \( x \) belongs to the grammar \( G \). The idea is similar to the CYK algorithm, but is simpler. Choose non-deterministically a production rule and a place in \( x \) on which to apply it. Rewrite the tape reversing the production rule i.e generate a string \( x' \) such that the chosen production rule applied on the \( x' \) at the chosen place, gives \( x \). Go on doing this, until we get to the start symbol of the grammar. We do not use extra tape because the CSG production rules are always non-decreasing in length, hence when we apply them in reverse, they are a series of operations that are non-increasing in length of the string.

For the other direction, i.e representing the LBA \( M \) using a grammar \( G \), intuitively, one has to represent the production rules themselves. We introduce nonterminals in
the grammar of the form $[\cdot, \cdot]$ where the first entry will hold the terminal string, and the second entry the LBA tape on which the simulation will be run. We need to take care of the end-symbols, as we cannot “erase” non-terminals in a CSG. We first generate the entire string using the following productions, including the end-markers. Below, $a$ represents all the terminals in the grammar.

\[
\begin{align*}
S_1 & \rightarrow [a, q_0 \vdash a]S_2 \\
S_1 & \rightarrow [a, q_0 \vdash a \sqsupset] \\
S_2 & \rightarrow [a, a]S_2 \\
S_2 & \rightarrow [a, a \sqsupset]
\end{align*}
\]

Now the productions representing the transition function.

\[
\begin{align*}
q[a, X] & \rightarrow [a, Y]p \text{ if } \delta(q, X) = (p, Y, R) \\
[a, X]q[b, Y] & \rightarrow p[a, X][b, Z] \text{ if } \delta(q, Y) = (p, Z, L)
\end{align*}
\]

Finally, once we reach a final state, we just spew off the string itself. Below, $q$ is a final state and $\alpha, \beta$ stand for any end-marker, non-terminal, tape symbol.

\[
\begin{align*}
[a, \alpha q \beta] & \rightarrow a \\
[a, a]b & \rightarrow ab \\
b[a, \alpha] & \rightarrow ba
\end{align*}
\]