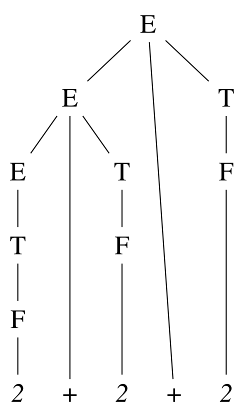
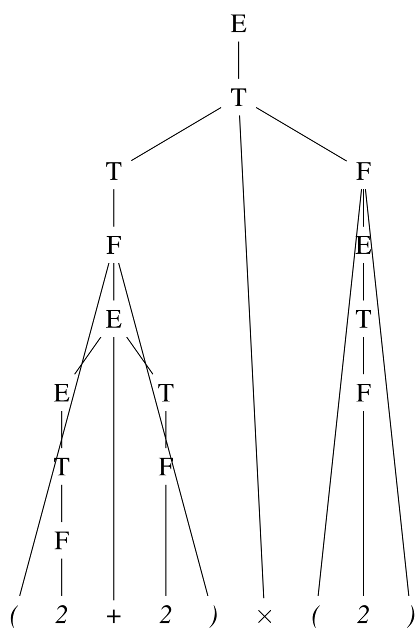
CMSC 303 Introduction to Theory of Computation, VCU
Assignment: 4
Name: Steven Hernandez

1. (a) 

   (b) 

(c) 

2. (a)

   (b)

3. (a)

$$\begin{array}{lcl} S & \rightarrow & 0 \mid 1 \mid 0T0 \mid 1T1 \\ T & \rightarrow & \epsilon \mid 1T \mid 0T \end{array}$$

Trivially 0 and 1 match. $0T0$ $1T1$ ensure that the first and last symbol are the same before moving past S into T. T simply allows you to add any symbols $\in \Sigma_{epsilon}$ recursively within the string obtained above.

(b)

$$\begin{array}{lcl} S\_0 & \rightarrow & 0 \mid 1 \mid 0S\_\{odd\} \mid 1S\_\{odd\} \\ S\_\{odd\} & \rightarrow & \epsilon \mid 0S\_\{even\} \mid 1S\_\{even\} \\ S\_\{even\} & \rightarrow & 0 \mid 1 \mid 0S\_\{odd\} \mid 1S\_\{odd\} \end{array}$$

Think of the labels such that S_{odd} means we have an odd length currently, thus we can only add $epsilon$ of one symbol, which then means we now have an even number of symbols (thus the S_{even}).

(c)

$$S \rightarrow \epsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$$

Unlike a, the only variable is S, this is because each time we recurse, we want to ensure whatever the sub-string contains, it always begins and ends with the same symbol, thus maintaining the palindrome.

(d)

$$S \rightarrow S$$

The grammar continues recursively forever. Never reaching only terminals, thus never reaching an accept state.

(e)

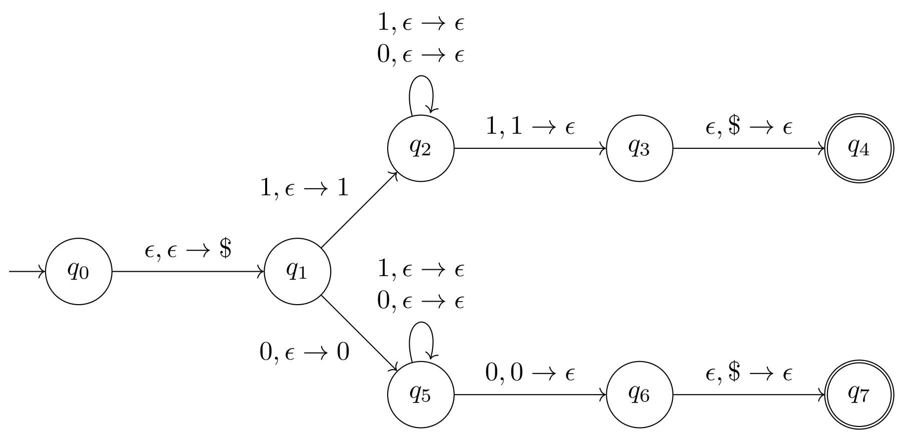
$$\begin{array}{lcl} S & \rightarrow & X\$C\$X \mid C\$X \mid X\$C \mid C \\ C & \rightarrow & 1C1 \mid 0C0 \mid \$ \mid \$X\$ \\ X & \rightarrow & \$X \mid 1X \mid 0X \mid \epsilon \end{array}$$

The idea for this grammar is that C always builds a palindrome. Note how from C, we either recursively wrap C with 0 or 1. After which, we can leave \$ in the center.

From the first step, if there are any symbols to the left or right of C, we delimite it with a \$. This keeps the grammar matching the language.

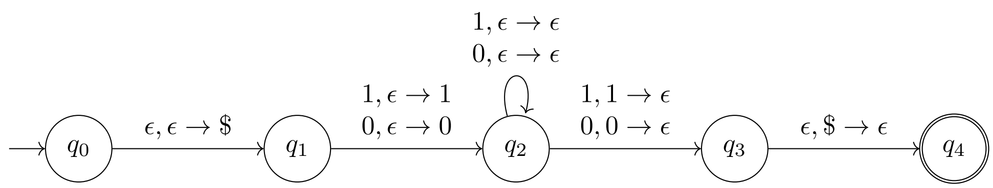
This produces the language when $i = n$ and $j = n + 1$, however does not account for $i = n$ and $j = n + t$ where $t > 1$. So, notice $C \to \$X\$$. This allows the palindrome we were building to have non-palindrome-like items in the middle of this palindrome. Notice though, that these symbols are delimited by \$ so that we keep separate the palindrome from earlier.

4. (a)



Notice however, this solution does not require a stack. The language could be modeled easily by an NFA with two branches as seen above.

Instead, a PDA where the stack is required, can be modeled as such:



(b)

(c)

(d)

5.