1. Let $L$ be the language with definition

\[ L = \{a^ib^k \mid 0 \leq i \leq k \leq 2i \}. \]

Give a recursive definition of $L$. Find $L_3$ according to your definition. Finally give a context free grammar which realizes $L$.

Solution:

**BASIS:** $\lambda \in L$

**RECURSIVE STEP:** $u \in L \Rightarrow aub, aub^2 \in L$.

**CLOSURE:** All elements in the language can be obtained from the basis with a finite number of applications of the recursive step.

With this definition:

$L_0 = \{\lambda\}$.
$L_1 = \{\lambda, ab, ab^2\}$.
$L_2 = \{\lambda, ab, ab^2, a^2b^2, a^2b^3, a^2b^4\}$.
$L_3 = \{\lambda, ab, ab^2, a^2b^2, a^2b^3, a^2b^4, a^3b^3, a^3b^4, a^3b^5, a^3b^6\}$.

\[ G : S \Rightarrow aSb \mid aSb^2 \]
2. Let $L$ be given by
   BASIS: $b, b^2, b^3 \in L$
   RECURSIVE STEP: If $u \in L$ then $auc \in L$ and $babuc^3 \in L_2$.
   CLOSURE: A string is in $L$ if it can be obtained from the basis by a finite number of
   applications of the recursive step.
   Prove by induction that for all $w \in L$, $n_a(w) + n_c(w)$ is even.

SOLUTION:
Proof. We will prove the statement by induction on the number of steps in the recursive
definition of $w \in L$.
Base Case: Suppose $w \in L_0$. Then $w \in \{b, b^2, b^3\}$, so $n_a(w) = n_c(w) = 0$, and
$n_a(w) + n_c(w) = 0$ which is even.
Inductive step. Suppose $n_a(u) + n_c(u)$ is even for all $u \in L_n$. Let $w \in L_{n+1}$, so either
$w = auc$ or $w = auc^3$.
   If $w = auc$ then $n_a(w) + n_c(w) = n_a(u) + 1 + n_c(u) + 1 = (n_a(u) + n_c(u)) + 2$ which is
even by the induction hypothesis.
   If $w = auc^3$ then $n_a(w) + n_c(w) = n_a(u) + 1 + n_c(u) + 3 = (n_a(u) + n_c(u)) + 4$ which is
even by the induction hypothesis.
In either case $n_a(w) + n_c(w)$ is even, which completes the induction step.
So the statement is proved by induction.
3. Give regular expressions for each of the following subsets of \(\{a, b, c\}^*\).
   a) Strings which do not contain the substrings \(aa\) or \(bb\).

   **SOLUTION:**
   Without \(c\) as a separator you have even or odd alternating \(a\)'s and \(b\)'s, so you could take \([((ab)^*(a \cup \lambda)] \cup [(ba)^*(b \cup \lambda)]\).
   With \(c\), you can take
   \[c^*\{[[ab]^*(a \cup \lambda)] \cup [(ba)^*(b \cup \lambda)]\}^+\{[[ab]^*(a \cup \lambda)] \cup [(ba)^*(b \cup \lambda)]\}\]
   Note this also matches the empty string.

   b) Set of strings of odd length containing exactly two \(b\)'s.

   **SOLUTION:** It seems simplest to think of four cases, where the three strings separate by the two \(b\)'s have length even-even-odd, even-odd-even, odd-even-even, or odd-odd-odd
   \[\left[\left[\left[\left[(a \cup c)^2\right]^*\left[b\left[\left[(a \cup c)^2\right]^*\left[b\left[\left[(a \cup c)^2\right]^*\left[(a \cup c)\right]\right]\right]\right]\right]\right]\right]\]
   \[\cup\left[\left[\left[\left[\left[(a \cup c)^2\right]^*\left[b\left[\left[(a \cup c)^2\right]^*\left[b\left[\left[(a \cup c)^2\right]^*\left[(a \cup c)\right]\right]\right]\right]\right]\right]\right]\right]\]
   \[\cup\left[\left[\left[\left[\left[(a \cup c)^2\right]^*\left[(a \cup c)\right]\right]\right]\right]\right]\]
   \[\cup\left[\left[\left[\left[\left[(a \cup c)^2\right]^*\left[(a \cup c)\right]\right]\right]\right]\right]\]

   c) Set of strings with an even number of \(a\)'s

   **SOLUTION:**
   \[\left[(a(b \cup c)^*a) \cup b \cup c\right]^*\]
4. Give a deterministic finite automaton which realizes each of the following languages on \{a, b, c\}.
   
a) Strings which do not contain the substrings \textit{aa} or \textit{bb}.

\begin{center}
\includegraphics[width=\textwidth]{image1.png}
\end{center}

b) Set of strings of odd length containing exactly two \textit{b}'s.

\begin{center}
\includegraphics[width=\textwidth]{image2.png}
\end{center}

c) Set of strings with an even number of \textit{a}'s

\begin{center}
\includegraphics[width=\textwidth]{image3.png}
\end{center}
5. Construct a context free grammar whose language is \( L = \{a^m b^i a^n \mid i = m + n \} \).

Prove that your grammar is correct.

SOLUTION:

\[
G : \quad S \rightarrow AB \\
A \rightarrow aAb \mid \lambda \\
B \rightarrow bBa \mid \lambda
\]

We will show that the sentential forms of the grammar, is equal to the set
\[ X = \{ S, a^m Ab^m B a^n, a^m Ab^m b^n a^n, a^m b^m b^n B a^n, a^m b^m b^n a^n \} \] with \( n, m \geq 0 \).

Let \( S \) Proof: \( S(G) \subseteq X \). Induction on the number of rules applied.

Base Case: If 0 rules have been applied, the sentential form is \( S \), which is in \( X \) as required.

Inductive Step. Assume that, after \( n \) rules have been applied, the sentential form \( n \) is in \( X \). If a rule applies \( w \) has an \( S \), and \( A \) or a \( B \).

If \( w \) has an \( S \), then \( w = S \) and the applying the only rule gives \( AB \in X \).

If an \( A \) rule applies then \( w \) is \( a^m Ab^m B a^n \) or \( a^m Ab^m b^n a^n \) and applying the rule gives \( a^{m+1} Ab^{n+1} B a^n \), \( a^{m+1} Ab^{n+1} b^n a^n \), \( a^m b^{m+1} B a^n \) or \( a^m b^{m+1} b^n a^n \), each of which are in \( X \).

If a \( B \) rule applies then \( w \) is \( a^m Ab^m B a^n \) or \( a^m b^m B a^n \) and applying the rule gives \( a^m Ab^m b^{n+1} Ba^n \), \( a^m b^{m+1} B a^n \), \( a^m Ab^m b^n a^n \), \( a^m b^m b^n B a^n \), or \( a^m b^m b^n a^n \), each of which are in \( X \).

So in any case, applying a rule gives a new rule in \( X \) and by induction \( L(G) \subseteq X \).

To show \( X \subseteq L(G) \), we provide a derivation sequence for the elements of \( X \).

\[
S \Rightarrow AB \Rightarrow a^m Ab^m B \Rightarrow a^m Ab^m b^n B a^n \Rightarrow a^m b^m b^n a^n
\]

Are derivation sequences for \( a^m Ab^m B a^n \), \( a^m b^m b^n B a^n \) and \( a^m b^m b^n a^n \) and

\[
S \Rightarrow AB \Rightarrow a^m Ab^m B \Rightarrow a^m Ab^m b^n B a^n \Rightarrow a^m Ab^m b^n a^n
\]

are is a derivation sequence for \( a^m Ab^m b^n a^n \).

So \( X \) is the set of sentential forms, and the the elements with no variables are the language \( L \), so \( L = L(G) \).
6. Let $G$ be the grammar given by

$$
G : S \rightarrow aAbB \mid ABC \mid a \\
A \rightarrow aA \mid a \\
B \rightarrow bBcC \mid b \\
C \rightarrow abc \\
D \rightarrow aAbBcCCC \mid AaA \mid E \\
E \rightarrow DE \mid ED
$$

Convert this grammar to Chomsky Normal Form.

First note that the final two symbols are useless, and can be eliminated. $D$ is unreachable and $E$ is non-terminable.

The grammar is non-contracting, and removing $D$ removes the only chain rule. If you keep $D$ and $E$ this chain rule must be removed. We will follow the procedure we covered in class: We first introduce $C_a$, $C_b$, $C_c$

$$
G : S \rightarrow C_aAC_bB \mid ABC \mid a \\
A \rightarrow C_aA \mid a \\
B \rightarrow C_bBC_cC \mid b \\
C \rightarrow C_aC_bC_c \\
C_a \rightarrow a \\
C_b \rightarrow b \\
C_c \rightarrow c
$$

and then split all the long rules:

$$
G : S \rightarrow C_aF_1 \mid AG_1 \mid a \\
A \rightarrow C_aA \mid a \\
B \rightarrow C_bH_1 \mid b \\
C \rightarrow C_aG_1 \\
F_1 \rightarrow AF_2 \\
F_2 \rightarrow C_bB \\
G_1 \rightarrow BC \\
H_1 \rightarrow BH_2 \\
H_2 \rightarrow C_cC \\
G_1 \rightarrow C_bC_c \\
C_a \rightarrow a \\
C_b \rightarrow b \\
C_c \rightarrow c
$$
7. a) Let $G$ be the grammar given by

\[
G : S \rightarrow AB \mid C \\
A \rightarrow aA \mid B \\
B \rightarrow bB \mid C \\
C \rightarrow cC \mid A \mid a
\]

Construct an equivalent grammar which does not contain chain rules.

We have seen this silly example before. $CHAIN(S) = \{S, A, B, C\}$, $CHAIN(A) = \{A, B, C\}$, $CHAIN(B) = \{A, B, C\}$, $CHAIN(C) = \{A, B, C\}$.

So $A$, $B$, and $C$ are all equivalent.

We can either simplify to

\[
G : S \rightarrow AB \mid aA \mid aB \mid aC \mid a \\
A \rightarrow aA \mid aB \mid aC \mid a \\
B \rightarrow aA \mid aB \mid aC \mid a \\
C \rightarrow aA \mid aB \mid aC \mid a
\]

or just

\[
G : S \rightarrow AA \mid aA \mid a \\
A \rightarrow aA \mid a
\]

b) For the grammar

\[
G : S \rightarrow AB \mid BCS \\
A \rightarrow aA \mid C \\
B \rightarrow bbB \mid b \\
C \rightarrow cC \mid \lambda
\]

construct an essential non-contracting grammar $G_L$ with non-recursive start

For the non-recursive start we “step back” to a new start symbol $S'$, and introduce a new chain rule. (The directions to require us to remove the chain rules.)

\[
G : S' \rightarrow S \\
S \rightarrow AB \mid BCS \\
A \rightarrow aA \mid C \\
B \rightarrow bbB \mid b \\
C \rightarrow cC \mid \lambda
\]
\( \text{Null}(G) = \{A, C\} \) so we translate by

\[
G : S' \quad S \\
S \rightarrow AB | BCS | B | BS \\
A \rightarrow aA | C | a \\
B \rightarrow bbB | b \\
C \rightarrow cC | c
\]

If you also remove the chain rules, \( CHAIN(S') = \{S', S, B\} \), \( CHAIN(S) = \{S, B\} \), \( CHAIN(A) = \{A, C\} \), \( CHAIN(B) = \{B\} \), and \( CHAIN(C) = \{C\} \). and we get

\[
G : S' \quad AB | BCS | bbB | b | BS \\
S \rightarrow AB | BCS | bbB | b | BS \\
A \rightarrow aA | cC | c | a \\
B \rightarrow bbB | b \\
C \rightarrow cC | c
\]

Both the pervious two grammars solve the problem. And any essentially non-contracting grammar giving the same language does as well, since the question did not specify how it was to be created.
8. Find an equivalent Deterministic Finite Automaton. (Note that in the diagram, \( \lambda \) is denoted by \( l \).)

![Automaton Diagram]

Also give a regular expression for the language.

We first compute the transition function:

<table>
<thead>
<tr>
<th></th>
<th>( \lambda ) – closure</th>
<th>( a )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>{( q_0, q_1, q_2, q_3, q_5 )}</td>
<td>{( q_2, q_3, q_4, q_5 )}</td>
<td>{( q_3, q_5 )}</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>{( q_1 )}</td>
<td>{( q_3 )}</td>
<td>{( q_3 )}</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>{( q_2, q_3, q_5 )}</td>
<td>{( q_2, q_3, q_4, q_5 )}</td>
<td>{( q_3, q_5 )}</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>{( q_3 )}</td>
<td>{( q_3 )}</td>
<td>{( q_3 )}</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>{( q_4 )}</td>
<td>{( q_4 )}</td>
<td>{( q_4 )}</td>
</tr>
<tr>
<td>( q_5 )</td>
<td>{( q_5 )}</td>
<td>{( q_5 )}</td>
<td>{( q_5 )}</td>
</tr>
</tbody>
</table>

Giving the automaton.

![Automaton Diagram]

For the regular expression we can use the expression graphs, and the regular expression easily simplifies to \( b^+ \cup a(a \cup b)^* \).