1-2. Transform, step by step, the grammar with rules $S \rightarrow \varepsilon$, $S \rightarrow aSb$, and $S \rightarrow bSa$ to Chomsky normal form. Show how the word baba will be generated in the resulting Chomsky-normal-form grammar.

<table>
<thead>
<tr>
<th>Step</th>
<th>Grammar</th>
<th>Tracing baba</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$S \rightarrow S$</td>
<td>$s_0$</td>
</tr>
<tr>
<td>1</td>
<td>$S \rightarrow aSb$</td>
<td>$V_a \rightarrow a$</td>
</tr>
<tr>
<td></td>
<td>$S \rightarrow bSa$</td>
<td>$V_b \rightarrow S$</td>
</tr>
<tr>
<td></td>
<td>$S \rightarrow \varepsilon$</td>
<td>$V_a \rightarrow a$</td>
</tr>
<tr>
<td>2</td>
<td>$S \rightarrow ab$</td>
<td>$S \rightarrow aV_a$</td>
</tr>
<tr>
<td></td>
<td>$S \rightarrow ba$</td>
<td>$S \rightarrow bV_b$</td>
</tr>
<tr>
<td></td>
<td>$S \rightarrow \varepsilon$</td>
<td>$S \rightarrow \varepsilon$</td>
</tr>
<tr>
<td>3</td>
<td>$S \rightarrow bS_a$</td>
<td>$S \rightarrow aV_a$</td>
</tr>
<tr>
<td></td>
<td>$S \rightarrow \varepsilon$</td>
<td>$S \rightarrow \varepsilon$</td>
</tr>
<tr>
<td>4</td>
<td>$S \rightarrow aS_a$</td>
<td>$S \rightarrow aV_a$</td>
</tr>
<tr>
<td></td>
<td>$S \rightarrow \varepsilon$</td>
<td>$S \rightarrow \varepsilon$</td>
</tr>
</tbody>
</table>

The tree for baba:

```
     V_a
    / \  \
   V_b  V_a
   / \  /  \  \n  b   V_a V_b
      / \  /  \  \n     V_a V_b a
```

file:///Q:/cs3350.18a/test3.html 11/8/2018
3-4. Use a general algorithm for transforming PDA into CFG to design a CFG that corresponds to the following pushdown automaton. This automaton has two states: the starting 0-state $s_0$ and the 1-state $s_1$. Both states are final. The transitions are:

- From $s_0$ to $s_0$, the transition is $0$, $\epsilon \rightarrow y$, for some symbol $y$
- From $s_0$ to $s_1$, the transition is $1$, $y \rightarrow \epsilon$.
- From $s_1$ to $s_1$, the transition is $1$, $y \rightarrow \epsilon$.

Show, step by step, how the word 01 will be generated by the resulting grammar.

```
1) A_{00} \rightarrow \epsilon
   A_{11} \rightarrow \epsilon

3) \[\begin{array}{c}
0, \epsilon \rightarrow y \\
1, y \rightarrow \epsilon \\
\end{array}\]

A_{01} \rightarrow O A_{00} 1

A_{01} \rightarrow O A_{01} 1

Tracing 01

S
A_{01}
O A_{00} 1
\epsilon
```
5. Use the general stack-based algorithms to show:
   - how the compiler will transform the expression \((1 - 10) / (8 - 11)\) into postfix form, and
   - how it will compute the value of the resulting postfix expression.

\[
\begin{array}{c}
\text{(1 - 10) / (8 - 11)} \\
\hline
1 \quad 10 \quad - \quad 8 \quad 11 \\
\hline
-1
\end{array}
\]
6. Illustrate the pumping lemma for context-free grammars by showing how it will represent the word \( w = +1.00 \) which is generated by the CFG with starting variable \( V \) and rules \( V \rightarrow SN.N, V \rightarrow N.N, S \rightarrow +, S \rightarrow -, N \rightarrow DN, N \rightarrow D, D \rightarrow 0 \), and \( D \rightarrow 1 \) as \( uvxyz \). Show, step-by-step, how the corresponding word \( uvvxyyz \) can be derived from this CFG.

\[
V \rightarrow SN.N \\
V \rightarrow N.N \\
S \rightarrow + \\
S \rightarrow - \\
N \rightarrow DN \\
N \rightarrow D \\
D \rightarrow 0 \\
D \rightarrow 1
\]

\[
\begin{align*}
u &= +1.0 \\
v &= 0 \\
x &= 0 \\
y &= \wedge \\
z &= \wedge
\end{align*}
\]

**Tracing/Deriving** \( uvvxyyz \)

\[
uvvxyyz = +1.000
\]
7. Prove that the language of all the words of the type $a^n b^n c^n d^n$, $n = 0, 1, 2, \ldots$, is not context-free.

Assume $L$ is CFG. Then for every $p$, let's take

$w = a^p b^p c^p d^p = a_1 a_2 \ldots b_1 b_2 \ldots c_1 c_2 \ldots d_1 d_2 \ldots$

so that $\text{len}(w) = 4p$.

Thus for every $u, v, x, y, z$ such that $w = uvxyz$ and $\text{len}(vxy) \leq p$, so we have one of the following cases:

1) $vxy$ is in $a^*$, so by pumping lemma we add $a^*_i$ but not $b^*_i, c^*_i, d^*_i$, so $uvu^i vxy z \notin L$.

2) $vxy$ is in $a^*$ and $b^*$, so by pumping lemma, we add $a^*_i$ & $b^*_i$, but not $c^*_i, d^*_i$, so $uvu^i vxy z \notin L$.

3) $vxy$ is in $b^*$, so by pumping lemma, we add $b^*_i$, but not $a^*_i, c^*_i, d^*_i$, so $uvu^i vxy z \notin L$.

4) $vxy$ is in $b^*$ and $c^*$, so by pumping lemma, we add $b^*_i & c^*_i$, but not $a^*_i, d^*_i$, so $uvu^i vxy z \notin L$.

5) $vxy$ is in $c^*$, so by pumping lemma, we add $c^*_i$, but not $a^*_i, b^*_i, d^*_i$, so $uvu^i vxy z \notin L$.

6) $vxy$ is in $c^*$ and $d^*$, so by pumping lemma, we add $c^*_i & d^*_i$, but not $a^*_i, b^*_i$, so $uvu^i vxy z \notin L$.

7) $vxy$ is in $d^*$, so by pumping lemma, we add $d^*_i$, but not $a^*_i, b^*_i, c^*_i$, so $uvu^i vxy z \notin L$.

In all possible cases, $uvu^i vxy z$ is not in the language, so there is a contradiction in our assumption that $L$ is CFG, therefore, $L$ is not CFG.
8. Design a Turing machine that, given a positive unary number \( n \), adds 2 to this number. Test it, step-by-step, on the example of \( n = 0 \).

\[
\begin{align*}
start, \ 1 & \rightarrow R, \ work 0 \\
work 0, \ 1 & \rightarrow R \\
work 0, \ \_ & \rightarrow 1, \ work 1, R \\
work 1, \ \_ & \rightarrow 1, \ back, \ L \\
back, \ 1 & \rightarrow L \\
back, \ \_ & \rightarrow halt
\end{align*}
\]
9. (For extra credit) Design a Turing machine that, given two unary numbers, computes their sum. The input is represented as two numbers separated by blank space. Test it, step-by-step, on the example of 2 + 2.

```
start, _ -> R, work
work, 1 -> R
work, 0, _ -> 1, add, R
add, 1 -> R
add, _ -> subtract, L
```

```
subtract, 1 -> _, back, L
back, 1 -> L
back, _ -> halt
```

![Turing machine diagrams](image-url)
10. (For extra credit) Let us consider possibly signed binary integers. Such numbers can be described by the following finite automaton. This automaton has a starting state s, an intermediate state i, a final state f, and a sink state k, and the following transitions:

- from s, + or − lead to i, while 0 or 1 lead to f;
- from i, 0 or 1 lead to f, while + or − leads to sink;
- from f, 0 or 1 lead to f, while + or − lead to sink.

Use the general algorithm to transform this finite automaton into a Turing machine. Show, step-by-step, how your Turing machine will accept the word +01.