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.

$$
\begin{align*}
S & \rightarrow \varepsilon \\
S & \rightarrow aSb \\
S & \rightarrow bSa \\
pre: & \quad S \rightarrow S
\end{align*}
$$

Step 0: $S \rightarrow ab$
$S \rightarrow ba$
$S_0 \rightarrow \varepsilon$

Step 1:
$S_0 \rightarrow aSb$
$S_0 \rightarrow bSa$
$S_0 \rightarrow ab$
$S_0 \rightarrow ba$

Step 2:
$V_a \rightarrow a$
$V_b \rightarrow b$
$S_0 \rightarrow V_aV_b$
$S_0 \rightarrow V_bV_a$
$S \rightarrow V_aSb$
$S \rightarrow V_bSa$
$S \rightarrow V_aV_b$
$S \rightarrow V_bV_a$
$S_0 \rightarrow \varepsilon$
$S_0 \rightarrow V_aV_b$
$S_0 \rightarrow V_bV_a$

Step 3:
$S \rightarrow V_aS V_b$, $V_a \rightarrow V_a S$
$S \rightarrow V_bS V_a$, $V_b \rightarrow V_b S$
$S_0 \rightarrow V_aS V_b$
$S_0 \rightarrow V_bS V_a$

With the above steps, the word baba can be generated as follows:

```
S \rightarrow V_bS V_a
S \rightarrow V_a V_b
S \rightarrow V_b V_a
S_0 \rightarrow \varepsilon
```

Diagram:

```
  S
 /|
V_a A
/ |\
V_b b
/  |
S_0
```

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

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

\[
(1 - 10) / (8 - 11)
\]
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.

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

$w = +1.00$

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

$u = +1.$
$y = 0$
$x = 0$
y = 0$
z = 0$

= +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.

$L = \{a^n b^n c^n d^n : n \geq 0\}$

By contradiction, let's assume $L$ is CFG. Then by pumping

**Lemma** \( \exists p \forall w \in L \) \((\text{length}(w) \geq p \rightarrow \exists u v x y z : w = u v x y z \text{ and length}(xy) \leq p \land \text{length}(v x y) \leq p \land \forall i (vv^i x y z \in L)) \)

Let's take \( w = a^p b^p c^p d^p \), its length \( (w) = 4p \geq p \), so we can represent it as \( uvxyz = a \ldots a b \ldots b c \ldots c d \ldots d \)

The fragment \( v x y \) cannot contain \( a \)'s, \( b \)'s, \( c \)'s and \( d \)'s because then \( \text{length}(v x y) \) is greater than \( p \) because all \( b \)'s are already length \( p \). We have the following options:

1. \( v x y \) is in \( a \)'s - we add \( a \)'s but not \( b \)'s, \( c \)'s or \( d \)'s : we now have more \( a \)'s than \( b \)'s, \( c \)'s, and \( d \)'s

2. \( v x y \) is in \( a \)'s and \( b \)'s - we add \( a \)'s and \( b \)'s but not \( c \)'s : we now have more \( a \)'s and \( b \)'s than \( c \)'s and \( d \)'s

3. \( v x y \) is in \( b \)'s - we add \( b \)'s but not \( a \)'s or \( c \)'s or \( d \)'s : we now have more \( b \)'s than \( a \)'s, \( c \)'s and \( d \)'s

4. \( v x y \) is in \( b \)'s and \( c \)'s - we add \( b \)'s and \( c \)'s but not \( a \)'s or \( d \)'s : we now have more \( b \)'s and \( c \)'s than \( a \)'s and \( d \)'s

5. \( v x y \) is in \( c \)'s - we add \( c \)'s but not \( a \)'s, \( b \)'s or \( d \)'s - we now have more \( c \)'s than \( a \)'s, \( b \)'s and \( d \)'s

6. \( v x y \) is in \( c \)'s and \( d \)'s - we add \( c \)'s and \( d \)'s but not \( a \)'s or \( b \)'s, we now have more \( c \)'s and \( d \)'s than \( a \)'s or \( b \)'s

7. \( v x y \) is in \( c \)'s - we add \( c \)'s but not \( a \)'s, \( b \)'s or \( c \)'s : so \( \not \exists u v^2 x y^2 z \in L \)

This is a contradiction and thus our assumption that \( L \) is CFG is false.

---

This is a contradiction and thus our assumption that $L$ is CFG is false.
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 \( n = 0 \).

Rules

- **start**, \( \uparrow \) \( \rightarrow \) working, R
- working, \( \uparrow \) \( \rightarrow \) working, R
- working, \( \downarrow \) \( \rightarrow \) add1, \( \rightarrow \) R
- add1, \( \uparrow \) \( \rightarrow \) back, \( \rightarrow \) R
- back, \( \uparrow \) \( \rightarrow \) back, R
- back, \( \downarrow \) \( \rightarrow \) halt

\( n = 0 \)
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 $2 + 2$.

**Rules**

- **Start**, $$ → num1, R
- num1, $$ → num1, R
- num1, $$ → num2, R
- num2, $$ → num2, R
- num2, $$ → erase, L
- erase, $$ → $$, add, L
- erase, $$ → back, L
- add, $$ → add, L
- add, $$ → $$, back, L
- back, $$ → back, L
- back, $$ → halt

---

$2 + 2$

```plaintext

\[\begin{array}{c}
\text{start} \\
\text{num1} \\
\text{num1} \\
\text{num2} \\
\text{num2} \\
\text{erase} \\
\text{add} \\
\text{add} \\
\text{back} \\
\text{back} \\
\text{halt}
\end{array}\]
```

---

file:///Q:/cs3350.18a/test3.html

11/8/2018
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 $+01$. 

\[ +01 \]

\[ \begin{array}{l}
\text{Start, } \uparrow \rightarrow s, R \\
 s, + \rightarrow i, R \\
 s, - \rightarrow i, R \\
 s, 0 \rightarrow f, R \\
 s, 1 \rightarrow f, R \\
i, 0 \rightarrow f, R \\
i, 1 \rightarrow f, R \\
i, + \rightarrow k, R \\
i, - \rightarrow k, R \\
\end{array} \]

\[ \begin{array}{l}
k, + \rightarrow k, R \\
k, - \rightarrow k, R \\
k, 0 \rightarrow k, R \\
k, 1 \rightarrow k, R \\
f, 0 \rightarrow f, R \\
f, 1 \rightarrow f, R \\
f, + \rightarrow k, R \\
f, - \rightarrow k, R \\
\end{array} \]

\[ \text{accept} \]

\[ \text{reject} \]