1. Describe context-free grammars for the following languages over the alphabet $\Sigma = \{0, 1\}$. For each non-terminal in your grammars, describe in English the language generated by that non-terminal.

(a) $\{0^a10^b10^c \mid a + b = c\}$

**Solution:** Any string $w = 0^a10^b10^{a+b} \in L$ can be decomposed into substrings as $w = 0^a1 \cdot (0^b10^b) \cdot 0^a$. The non-terminal $B$ generates the parenthesized substring $0^b10^b$; the starting non-terminal then generates the outer $0$s.

- $S \rightarrow 0S0 \mid 1B$
- $B \rightarrow 0B0 \mid 1$

**Rubric:** 2 points = 1 for grammar + 1 for justification.
(b) \( \{ w \in (0 + 1)^+ \mid \#(0, w) \leq 2 \cdot \#(1, w) \} \)

**Solution (dropping 0s):** Let \( L^\leq \) be the target language. We modify the grammar for the language \( L^\geq := \{ w \in (0 + 1)^+ \mid \#(0, w) = 2 \cdot \#(1, w) \} \) described in the lab solutions:

\[
S \to \epsilon \mid SS \mid 00S1 \mid 1S00 \mid 0S1S0
\]

Dropping any number of 0s from any string in \( L^\geq \) leaves a string in \( L^\leq \); moreover, *every* string in \( L^\leq \) can be obtained from some string in \( L^\geq \) by dropping 0s.

Thus, we can transform the grammar for \( L^\geq \) into a grammar for \( L^\leq \) by dropping 0s from the right side of every production in all possible ways:

\[
S \to \epsilon \mid SS \mid 00S1 \mid 0S1 \mid S1 \mid 1S00 \mid 1S0 \mid 1S \mid 0S1S0 \mid 0S1S \mid S1S
\]

The last three productions in this grammar are redundant, so we can remove them. For example, we don’t need the production \( S \to S1S \), because other productions give us the derivation \( S \Rightarrow SS \Rightarrow 1S \).

\[
S \to \epsilon \mid SS \mid 00S1 \mid 0S1 \mid S1 \mid 1S00 \mid 1S0 \mid 1S \mid 0S1S0
\]

If we add the trivial production \( S \to 1 \) (replacing the derivation \( S \Rightarrow 1S \Rightarrow 1 \)), we can remove two more redundant productions.

\[
S \to \epsilon \mid 1 \mid SS \mid 00S1 \mid 0S1 \mid 1S00 \mid 1S0 \mid 1S \mid 0S1S0
\]

\[\blacksquare\]

**Solution (0, -1, or less):** For any string \( w \), let \( \Delta(w) = \#(0, w) - 2 \cdot \#(1, w) \).

\[
L \to \epsilon \mid ZL \mid ML \mid 1L \quad \Delta \leq 0
\]

\[
Z \to \epsilon \mid ZZ \mid 00Z1 \mid 1Z00 \mid 0Z1Z0 \quad \Delta = 0
\]

\[
M \to ZM \midMZ \mid 0Z1 \mid 1Z0 \quad \Delta = -1
\]

The non-terminals in this grammar respectively generate the following languages.

\[
L = \{ w \in (0 + 1)^+ \mid \Delta(w) \leq 0 \} \quad \text{“Less or equal”}
\]

\[
Z = \{ w \in (0 + 1)^+ \mid \Delta(w) = 0 \} \quad \text{“Zero”}
\]

\[
M = \{ w \in (0 + 1)^+ \mid \Delta(w) = -1 \} \quad \text{“Minus one”}
\]

Our target language is \( L \). A grammar for \( Z \) appears in the Lab 4½ solutions. For any string \( w \), we have \( \Delta(w) = -1 \) (that is, \( w \in M \)) if and only if at least one of the following conditions holds:

- \( w \) has a non-empty proper prefix in \( Z \), and the rest of \( w \) is in \( M \).
- \( w \) has a non-empty proper prefix in \( M \), and the rest of \( w \) is in \( Z \).
- \( \Delta(x) > 0 \) for *every* non-empty proper prefix \( x \) of \( w \). Then \( w = 0z1 \) for some string \( z \in Z \).
• $\Delta(x) < -1$ for every non-empty proper prefix $x$ of $w$. Then $w = 1z\emptyset$ for some string $z \in Z$.

Finally, $\Delta(w) \leq 0$ if and only if at least one of the following conditions holds:

• $w$ is empty.
• $w$ has a non-empty prefix in $Z$, and the rest of $w$ is in $L$. (This includes the case $w \in Z$.)
• $w$ has a non-empty prefix in $M$, and the rest of $w$ is in $L$. (This includes the case $w \in M$.)
• $\Delta(x) \leq -2$ for every non-empty prefix $x$ of $w$. (In particular, $\Delta(w) \leq -2$.) Then $w = 1y$ for some string $y$ with $\Delta(y) \leq 0$.

Rubric: 4 points = 2 for grammar + 2 for justification. These solutions have more detail than necessary for full credit.
(c) Strings in which the substrings 00 and 11 appear the same number of times. For example, 1100011 ∈ L because both substrings appear once, but 01000011 \∉ L. [Hint: This is the complement of the language you considered in HW2.]

Solution (counting): Let \( (#(11, w) \) denote the number of times 11 appears as a substring of \( w \), and let \( #(1^*, w) \) denote the number of runs of 1s in \( w \). For example:

\[
#(11, 111100111101) = 5 \quad #(1^*, 111100111101) = 3
\]

Our grammar is based on the observation that each 1 in a binary string is the start of a 11 substring, except for the last 1 in every run; thus, symbolically, we have

\[
#(11, w) = #(1, w) - #(1^*, w).
\]

Symmetrically, \( #(00, w) = #(0, w) - #(0^*, w) \). But because runs of 0s and 1s in any binary string \( w \) alternate, \( #(0^*, w) \) and \( #(1^*, w) \) always differ by at most 1.

Further case analysis implies that \( L \) contains a binary string \( w \) if and only if one of the following conditions holds:

- \( w = \varepsilon \)
- \( w \) starts with 0 and ends with 1, and \( #(0, w) = #(1, w) \)
- \( w \) starts with 1 and ends with 0, and \( #(0, w) = #(1, w) \)
- \( w \) starts with 0 and ends with 0, and \( #(0, w) = #(1, w) + 1 \). (In this case, dropping the final 0 leaves a string with equal 0s and 1s.)
- \( w \) starts with 1 and ends with 1, and \( #(0, w) = #(1, w) - 1 \). (In this case, dropping the final 1 leaves a string with equal 0s and 1s.)

In the following grammar, each nonterminal \( \_E_\) generates strings that start with symbol \( a \), end with symbol \( z \), and have equal numbers of 0s and 1s. Missing subscripts indicate that we don’t care about the first and/or last symbol. The nonterminals \( \_E_0 \) and \( \_E_1 \) never appear on the right side of a production, so we can ignore them. In each case, we derive the production rules from the grammar for \( L(E) = \{ w \mid #(0, w) = #(1, w) \} \) described in the lecture notes.

\[
S \rightarrow \varepsilon \mid 0 \mid 1 \mid \_E 0 \mid \_E 1 \mid 1 E 0 \mid 1 E 1
\]

\[
E \rightarrow \varepsilon \mid EE \mid 0 E 1 \mid 1 E 0
\]

\[
\_E \rightarrow \_E E \mid 0 E 1
\]

\[
\_E \rightarrow \_E E \mid 1 E 0
\]

\[
E_0 \rightarrow EE_0 \mid 1 E 0
\]

\[
E_1 \rightarrow EE_1 \mid 0 E 1
\]

\[
\_E_0 \rightarrow \_E E_1 \mid 0 E 1
\]

\[
\_E_0 \rightarrow \_E E_0 \mid 1 E 0
\]

\[
\_E_1 \rightarrow \_E E_1 \mid 0 E 1
\]

\[
\_E_1 \rightarrow \_E E_0 \mid 1 E 0
\]
Solution (simpler counting): For any string $w$, let $\Delta(w) = \#(\emptyset, w) - \#(1, w)$. The case analysis in the previous solution implies that our target language $L$ contains a binary string $w$ if and only if one of the following conditions holds:
- $w = \varepsilon$
- $w$ starts with 0 and ends with 1, and $\Delta(w) = 0$
- $w$ starts with 1 and ends with 0, and $\Delta(w) = 0$
- $w$ starts with 0 and ends with 0, and $\Delta(w) = 1$
- $w$ starts with 1 and ends with 1, and $\Delta(w) = -1$

In the third case, either $w = \emptyset$ or $w = \emptyset x \emptyset$ for some string $x$ with $\Delta(x) = -1$. If $w = \emptyset x \emptyset$, then $\Delta(\emptyset) = +1$ and $\Delta(x) = 0$, so $w$ has a prefix with $\Delta = 0$, and the shortest such prefix must end with 1; it follows that $w = \emptyset y 1 z \emptyset$ where $\Delta(y) = 0$ and $\Delta(z) = 0$. Similar analysis applies to the fourth case above.

We conclude with the following grammar for $L$, where $E$ generates all strings with $\Delta = 0$.

$$S \rightarrow \varepsilon | 0 | 1 | 0 E 1 | 1 E 0 | 0 E 1 E 0 | 1 E 0 E 1$$

$$E \rightarrow \varepsilon | E E | 0 E 1 | 1 E 0$$

Solution (brute-force terminators): Sorry, this is really ugly, but it was the first solution I found.

Our grammar treats the last symbol in each run of 0s or 1s as a distinct symbol; we indicate this distinction by writing these symbols in blue with hats ($\hat{0}$ or $\hat{1}$). For example, we would write the string $11000111110111$ as $1\hat{1}000\hat{0}1111\hat{0}1111$. With this marking, every binary string matches the regular expression $(0^*\hat{0} + \varepsilon)(1^*\hat{1}0^*\hat{0})^*(1^*\hat{1} + \varepsilon)$.

Let $L'$ be the set of all strings in $\{0, 1, \hat{0}, \hat{1}\}^*$ that match this regular expression and have equal numbers of red 0s and red 1s. We construct a grammar for $L$ by first constructing a grammar for $L'$, and then ignoring the colors in the terminal alphabet (but not in the non-terminal names).

The following grammar generates $L'$, using a modification of the grammar $S \rightarrow \varepsilon | SS | 0 S 1 | 1 S 0$ for all binary strings with equal 0s and 1s. Specifically, each of the 16 non-terminals $S_{ab}$ generates the strings in $L'$ that start with $a$ and end with $b$.

$$S \rightarrow \varepsilon | S_{\emptyset \emptyset} | S_{\emptyset 1} | S_{1 \emptyset} | S_{1 1} | S_{\emptyset \hat{0}} | S_{\emptyset \hat{1}} | S_{\hat{0} \emptyset} | S_{\hat{0} \hat{1}} | S_{\hat{1} \emptyset} | S_{\hat{1} \hat{1}}$$

$$S_{\emptyset \emptyset} \rightarrow S_{\emptyset \hat{0}} | S_{\emptyset \hat{1}} | \hat{0}$$

$$S_{\emptyset 1} \rightarrow S_{\emptyset \hat{1}} | S_{\hat{0} \hat{1}}$$

$$S_{1 \emptyset} \rightarrow S_{\hat{0} \hat{0}} | S_{1 \hat{0}}$$

$$S_{1 1} \rightarrow S_{\hat{1} \hat{1}} | S_{1 \hat{0}}$$

$$S_{\hat{0} \emptyset} \rightarrow S_{\hat{0} \hat{0}} | S_{\hat{1} \hat{0}}$$

$$S_{\hat{0} \hat{1}} \rightarrow S_{\hat{0} \hat{0}} | S_{1 \hat{1}}$$

$$S_{\hat{1} \emptyset} \rightarrow S_{\hat{1} \hat{0}} | S_{\hat{1} \hat{0}}$$

$$S_{\hat{1} \hat{1}} \rightarrow S_{\hat{1} \hat{0}} | S_{\hat{1} \hat{0}}$$
In the first three groups of productions, we peel off as many blue bits as possible from either end of the string; most the real work is done in the last group. In the productions for $S_{00}$ and $S_{11}$, we reduce the number of cases by splitting off the shortest non-empty prefix with equal 0s and 1s.

**Rubric:** 4 points = 2 for grammar + 2 for justification. These solutions have more detail than necessary for full credit.
2. Let \( \text{inc}: \{0,1\}^* \rightarrow \{0,1\}^* \) denote the \textit{increment} function, which transforms the binary representation of an arbitrary integer \( n \) into the binary representation of \( n+1 \), truncated to the same number of bits.

Let \( L \subseteq \{0,1\}^* \) be an arbitrary regular language. Prove that \( \text{inc}(L) = \{\text{inc}(w) \mid w \in L\} \) is also regular.

\textbf{Solution (forward NFA)}: Let \( M = (Q,s,A,\delta) \) be a DFA that accepts \( L \). We construct an NFA \( M' = (Q',s',A',\delta') \) that accepts \( \text{inc}(L) \) as follows:

\[
Q' = Q \times \{0,1\} \cup \{s'\}
\]

\( s' \) is an explicit state in \( Q' \)

\[
A' = \{(q,1) \mid q \in Q\}
\]

\[
\delta(s', \varepsilon) = \{(s,0), (s,1)\}
\]

\[
\delta(s', a) = \emptyset \quad \text{for all } a \in \Sigma
\]

\[
\delta'(q,0,0) = \{(\delta(q,0),0)\} \quad \text{for all } q \in Q
\]

\[
\delta'(q,0,1) = \{(\delta(q,1),0), (\delta(q,0),1)\} \quad \text{for all } q \in Q
\]

\[
\delta'(q,1,0) = \{(\delta(q,1),1)\} \quad \text{for all } q \in Q
\]

\[
\delta'(q,b,\varepsilon) = \emptyset \quad \text{for all } q \in Q \text{ and } b \in \{0,1\}
\]

Our machine \( M' \) reads a string of the form \( x10^n \) or \( 0^n \) and passes the \textit{decremented} string \( x01^n \) or \( 1^n \) to \( M \). State \( (q,0) \) indicates that \( M \) is in state \( q \) and \( M' \) is reading the initial prefix \( x \); state \( (q,1) \) indicates that \( M \) is in state \( q \) and \( M' \) is flipping bits before passing them to \( M \). When \( M' \) begins or reads a \( 1 \), it guesses whether to switch from passing bits directly to \( M \) to inverting them.

\textbf{Rubric}: 10 points: standard language transformation rubric.

\textbf{Solution (reverse DFA)}: Recall from class that the reversal \( \text{rev}(L) = \{\text{rev}(w) \mid w \in L\} \) of any regular language \( L \) is regular.

For any string \( w \), define \( \text{linc}(w) := \text{rev}(\text{rev}(\text{inc}(w))) \), and for any language \( L \), define

\[
\text{linc}(L) := \{\text{linc}(w) \mid w \in L\} = \text{rev}(\text{rev}(\text{inc}(L))).
\]

The name \( \text{linc} \) is short for “left increment”; this is the increment function for binary strings whose least significant bits are on the left. For any integer \( n \geq 0 \) and any string \( x \), we have \( \text{linc}(1^n0x) = 0^n1x \) and \( \text{linc}(1^n) = 0^n \).

I claim that for any regular language \( L \), the language \( \text{linc}(L) \) is also regular. This claim immediately implies that \( \text{inc}(L) = \text{rev}(\text{linc}(\text{rev}(L))) \) is regular, solving the homework problem.

To prove my claim, let \( M = (Q,s,A,\delta) \) be an arbitrary DFA that accepts \( L \). We
construct a DFA \( M' = (Q', s', A', \delta') \) that accepts \( \text{linc}(L) \) as follows:

\[
Q' = Q \times \{0, 1\} \\
s' = (s, 1) \\
A' = A \times \{0, 1\}
\]

\[
\delta'((q, 1), 0) = (\delta(q, 1), 1) \quad \text{for all } q \in Q \\
\delta'((q, 1), 1) = (\delta(q, 0), 0) \quad \text{for all } q \in Q \\
\delta'((q, 0), 0) = (\delta(q, 0), 0) \quad \text{for all } q \in Q \\
\delta'((q, 0), 1) = (\delta(q, 1), 0) \quad \text{for all } q \in Q
\]

The transition function can be more concisely written as

\[
\delta'((q, b), a) = (\delta(q, a \oplus b), a \land b)
\]

for all \( q \in Q \) and all \( a, b \in \{0, 1\} \). Intuitively, our new machine \( M' \) reads a string of the form \( 0^n1x \) or \( 0^n \) and passes the decremented string \( 1^n0x \) or \( 1^n \) to \( M \). Each state \( (q, b) \) indicates that \( M \) is in state \( q \) and the current “borrow” bit is \( b \). Equivalently, \( b = 1 \) if and only if \( M' \) has not yet read a \( 0 \).  

**Rubric:** 10 points = 2 for defining \( \text{linc} \) + 5 points for \( \text{linc} \) transformation (standard language transformation rubric, scaled) + 1 for reversal transformation (from lecture/notes) + 2 for remaining details. No credit for assuming/building a DFA that “reads its input from right to left”; that’s not how DFAs are defined.
3. Prove that if $L$ and $L'$ are regular languages, then $\text{shuffles}(L, L')$ is also a regular language.

**Solution:** Let $L_1$ and $L_2$ be arbitrary regular languages. Let $M_1 = (Q_1, S_1, A_1, \delta_1)$ be an arbitrary DFA that accepts $L_1$, and let $M_2 = (Q_2, S_2, A_2, \delta_2)$ be an arbitrary DFA that accepts $L_2$. We build a new NFA $M = (Q, S, A, \delta)$ that accepts $\text{shuffles}(L_1, L_2)$ using a modified product construction:

$$Q = Q_1 \times Q_2$$
$$S = (s_1, s_2)$$
$$A = A_1 \times A_2$$
$$\delta((q_1, q_2), a) = \{(\delta_1(q_1, a), q_2), (q_1, \delta_2(q_2, a))\}$$

Intuitively, $M$ runs the given machines $M_1$ and $M_2$ simultaneously. At each step, $M$ nondeterministically chooses whether to pass the next input symbol to $M_1$ or to $M_2$. Finally, when the input is consumed, $M$ accepts if and only if both $M_1$ and $M_2$ are in accepting states.

**Rubric:** 10 points: standard language transformation rubric