Problem 2. We provide a context-free grammar \( G \) with \( L(G) = L \).

Since \( L \) is closed under "U", we consider the following two cases: when \( |u| \neq |v| \) and when \( |u| = |v| \).

Case when \( |u| \neq |v| \):

Use a variable \( T \) to generate an arbitrary string of equal-length sides (with a variable \( X \) generating a single arbitrary symbol), and then, use another variable \( U \) to generate an arbitrary string \( u \) of length \( \leq 1 \) and concatenate this string to either the beginning \((|u|> |v|)\) or the end \((|u|< |v|)\) of the string generated by \( X \).

\[
S_1 \rightarrow TU \mid UT \\
T \rightarrow XTX \mid \varepsilon \\
U \rightarrow XU \mid X \\
X \rightarrow 0 \mid 1
\]

Case when \( |u| = |v| \): With the idea learned in lecture "mismatched",

At corresponding positions

\[
\begin{array}{cccccccc}
\text{\\text{U}} & \text{\\text{}} & \text{\\text{}} & \text{\\text{}} & \text{\\text{}} & \text{\\text{}} & \text{\\text{}} & \text{\\text{}} \\
\text{\text{A}} & \text{\text{}} & \text{\text{}} & \text{\text{}} & \text{\text{}} & \text{\text{}} & \text{\text{}} & \text{\text{}} \\
\text{\text{B}} & \text{\text{}} & \text{\text{}} & \text{\text{}} & \text{\text{}} & \text{\text{}} & \text{\text{}} & \text{\text{}} \\
\end{array}
\]

Use two variables \( A \) and \( B \):

Variable \( A \) generates an equal number of symbols on either side of itself and then becomes "\( \varepsilon \)"

Similarly, for variable \( B \) - with a basis of becoming "\( \varepsilon \)"

So, now,

\[
S_2 \rightarrow AB \mid BA
\]

The production rules for \( A \) and \( B \) are similar to those for the variable \( T \) above.
Problem 3. Idea: Construct a pushdown automaton so that if the input \( w \) satisfies \( 2 \#_a(w) = 3 \#_b(w) \), the PDA will reject it. Otherwise, it will accept \( w \). The PDA will use its stack to keep track of how many 'a's or 'b's are "in extra" (for \( 2 \#_a(w) \) to match with \( 3 \#_b(w) \)).

To do so, the PDA assigns each 'a' to be worth 2 units, and each 'b' to be worth -3 units, and the stack keeps track of the "net total" as it processes the input string.

Check that \( w \) is accepted only if the net total after processing \( w \) is not a unit.

Problem 5
(a) \( L \) is not a CFL.

Suppose the contrary that \( L \) was CFL.

Let \( n \) be the pumping lemma constant.

Consider \( z = 0^{2n} \ 0^n \ 1^n \ 0^{2n} \in L \) with \( 121 \geq n \).

Consider all possible \( u, v, w, x, y \in \{0, 1\}^* \) such that \( z = uvwxy \)

such that

\[ |vwx| \leq n \]

\[ |v| \geq 1. \]
There are only three cases to consider:

Case 1: when \( v \cdot w \) contains only Os and some Os are chosen from the last \( 0^{2n} \) of \( z \).

Let \( i \) be an integer with \( 7n > 1 \cdot v \cdot w \cdot (i+1) \geq 6n \). Then, either the length of \( u \cdot w \cdot x \cdot y \cdot z \) is not a multiple of 3, or the string is of the form \( u \cdot u \cdot w \) such that \( |u| = |w| = |x| = 1 \). Hence, \( u \cdot u \cdot w \) consisting of all Os and hence \( u \cdot w \) consisting of not all Os \( \Rightarrow w \neq u \).

Case 2: when \( v \cdot w \) does not contain any Os in the last \( 0^{2n} \) of \( z \).

Then, either the length of \( u \cdot w \cdot x \cdot y \cdot z \) is not a multiple of 3, or the string is of the form \( u \cdot w \) such that \( |u| = |w| = 1 \). Hence, \( u \cdot w \) consisting of all Os and hence \( w \neq u \).

Case 3: when \( v \cdot w \) is not all Os and some Os are chosen from the last \( 0^{2n} \) of \( z \).

As \( 1 \cdot v \cdot w \cdot 1 \leq n \), \( v \cdot w \cdot x \) in this case must be a substring in \( 1^{2n} \). Then, either the length of \( u \cdot w \cdot x \cdot y \cdot z \) is not a multiple of 3, or the string is of the form \( u \cdot w \) such that \( |u| = |w| = 1 \). Hence, \( u \cdot w \) consisting of not all Os \( \Rightarrow w \neq u \).
(b) \( L_2 \) is not CF.

Suppose \( L_2 \) were CF.

Let \( n > 0 \) be the pumping lemma constant.

Consider \( z = 1^n 0^n 1^n \in L_2 \) with \( |z| \geq n \).

Consider all possible \( u, v, w, x, y \in \{0, 1\}^* \) with \( z = uvwx \)

\[ 1 \leq |vwx| \leq n \]
\[ |vx| \geq 1 \] .

Notice that \( u \) and \( v \) should not contain both 0 and 1, otherwise "pumping up" would be inconsistent with the ordering of the string \((u \# 0^n 1^n u \#)\). Consequently, \( v, x \in \{0^n 1^n\}^* \).

Clearly, \( u \) and \( x \) can not both be in \( 0^* \),

otherwise, only the make \( y \) of Os would be affected by pumping the string and the arithmetical relation between the numbers of Os and 1s would be violated.

Hence, there are only two major cases to consider:

Case when \((v \in \{0^* \} \& x \in \{1^* \})\) or \((v \in \{0^* \} \& x \in \{1^* \})\).

Consider the former subcase as the latter one can be proved with similar arguments.

If \( v \in \{0^* \} \& x \in \{1^* \} \), then the pumped string would be \( uv^n w^n x^k = 1^n 0^i 1^n + (k-1) 1^n v^n 1^n - x^n (0^n 1^n)^k \).

We can check that the only choice of \( i, k \)

\[ 1 \leq k \leq |x| > 0 \text{ so that } v^n w^n x^k \]

\[ k \geq 0 \text{ } (n-1)v^{(k-1)} v^n (n-1)1^{(k-1)} 1^n = n^2 \]

\[ 1 \leq |v|= |x| > 0 \text{ , a contradiction.} \]
Case when \( r, t \in 1^* \).

In such case \( r \) should be a subset of the first sequence \( 1 \). \( t \) is the second one.

Otherwise, the condition \( |vwxy| \leq n \) would not be satisfied. That is,

\[
\underbrace{11 \ldots 11}_{\text{\( n \) times}} \underbrace{11 \ldots 11}_{\text{\( n \) times}} \underbrace{00 \ldots 00}_{\text{\( n \) times}} 11 \ldots 11 \ldots 11
\]

Again, if we pump \( r \) at \( z \), we have

\[
u^k\overline{vwz^2}y = 1^n-1v^l+(k^l+1)v^l 0^n 1^{n^2-(k^l+1)}z^l,
\]

or the only feasible values for \( 1v^l \) and \( 1z^l \geq 0 \) for any choice of \( k \geq 0 \) are \( v^l = z^l = 0 \),

again a contradiction.

(c) \( L_3 \) is CF.

Idea: View any string in \( L_3 \) as concatenation of five substrings: \( AWBWC \), where

\( W \) is the reversal of \( W \), \( A \) can be any string in \( \{a, b, \#\}^* \) ending with \( \# \) or just \( \varepsilon \).

Similarly, \( C \) can be any string in \( \{a, b, \#\}^* \) starting with \( \# \) or just \( \varepsilon \). The "center" \( B \) can be any string starting or ending with \( \# \) or \( a \) or \( b \) or \( \varepsilon \), where the latter three cases correspond to the case where \( i = j \) (\( \alpha_i = \alpha_j \)).

Can you construct the underlying CFG?
Problem 7.

Non-shrinkable is Turing-decidable.

Construct a Turing machine \( M \) that decides Non-shrinkable as follows.

On input \( R \) of \( M \):
1. Check if \( R \) is a valid regular expression:
   - "true": go to step 2,
   - "false": \( M \) halts and rejects \( R \)
2. \( M \) constructs a DFA \( D \) for the language \( L(R) \) (run NFA and DFA in text or lecture notes)
3. \( M \) runs a depth-first search starting from \( q_{start} \) of \( D \), and removes all states in \( D \) that are not reachable from \( q_{start} \) from \( D \).
4. \( M \) for each accepting state \( q \) in \( D \), runs a depth-first search starting from \( q \) and check if another accepting state (not equal to \( q \)) is reachable from \( q \), or if there is a loop from \( q \) to itself.
   - If any such paths or loops are found,
     - \( M \) halts and rejects \( R \).
   - Otherwise, \( M \) halts and accepts \( R \).

Note: It is first required to remove all the states (actually, just accepting states) not reachable from \( q_{start} \), since these states cannot lead to any
Problem 8
Non-Empty is Turing-recognizable.

We proceed as in the construction of an equivalent TM $N$ from a given TM $M$:

On input $w$ to $N$:
1. Check if $w$ is of the form $\langle M \rangle$ for some TM $M$:
   "true": go to step 2
   "false": $N$ halts and rejects $w$
2. loop for $i = 1, 2, \ldots$
   $N$ simulates $M$ on all strings of length at most $i$ for $i$ steps (for each such string)
   if $M$ accepts some string,
   then $N$ halts and accepts $w$
   end loop

Note: If $L(M) \neq \emptyset$, we can see that for some $i$, $N$ will halt and accept its input.
Problem 9.
(a) Suppose that $L$ were Turing-decidable. We can construct an “algorithm” for deciding $\text{Halt}_{TM}$ — which will be a contradiction.

Given an input $<M, w>$, we want to decide if $M$ halts on $w$. We first construct a TM $N$, which just ignores its input and simulates $M$ on $w$. Hence, $N$ will halt on $E$ if and only if $M$ halts on $w$.

Let $n$ be the number of states in $N$. We can now test if $N$ halts on $E$ as follows:

$k := 1$;

while (true) loop
    if $(n, k) \in L$
        break
    else
        $k := k + 1$

end loop;

run $N$ on $E$ for $k$ steps

halt and accept if $N$ halts in at most $k$ steps
else reject

Since the number of $n$-state TMs is finite, there must be some maximum $k$ such that all such TMs either halt in $k$ steps or run forever.
The above algorithm first finds this $k$, and then simply checks if $N$ halts in $k$ steps.
(b) We show that \( L \) is Turing-recognizable.

Since \( L \) is not Turing-decidable, this implies that \( L \) cannot be Turing-recognizable.

\[
L = \{ (n, m) \mid \text{some } n \text{-state TMs halt on } \varepsilon \text{ after more than } m \text{ steps} \}
\]

Since there are only a finite number of TMs with \( n \) states, we can simulate all of them in parallel on \( \varepsilon \).

If \( (n, m) \in L \) then at least one of the TMs will halt after more than \( m \) steps, and we will halt and accept.