Please note that there is more than one way to answer most of these questions. The following only represents a sample solution.

Problem 1: Linz 3.1.21 and 3.2.18

3.1.21: Give a general method by which any regular expression \( r \) can be changed into \( \hat{r} \) such that \((L(r))^R = L(\hat{r})\).

We will give a couple methods. The first uses only regular expressions (RE). The second convert the regular expression to an NFA, finds the reverse and then converts that back into a regular expression.

Method 1. Given a RE, \( r \), the following procedure \( r^R \) will compute the reverse of \( r \):

1. If \( r = \emptyset \), then \( r^R = \emptyset \).
2. If \( r = \lambda \), then \( r^R = \lambda \).
3. If \( r = a \) for some \( a \in \Sigma \), then \( r^R = a \).
4. If \( r = r_1 + r_2 \), for REs \( r_1 \) and \( r_2 \), then \( r^R = r_1^R + r_2^R \).
5. If \( r = r_1^* \), for some RE \( r_1 \), then \( r^R = (r_1^R)^* \).

Now we will show our procedure is correct. We will proceed by induction on the length of the RE (You are not required to show induction on the homework, but should give and explanation why the method would halt and why each part is correct).

- **Base case:** If \(|r| = 1\), then \( r \) is one of the following forms:
  1. \( r = \emptyset \), then clearly \( r^R = \emptyset \).
  2. \( r = \lambda \), then clearly \( r^R = \lambda \).
  3. \( r = a \) for some \( a \in \Sigma \), then clearly \( r^R = a \).

- **Induction Hypothesis:** Assume for all \(|r| < n\) that \((L(r))^R = L(\hat{r})\).

- **Now, let \(|r| = n > 1\). Then \( r \) has one of the following forms:
4. \( r = r_1 + r_2 \), for REs \( r_1 \) and \( r_2 \). Note that \( |r_1| < n \) and \( |r_2| < n \). Now \( w^R \in (L(r))^R = (L(r_1 + r_2))^R = (L(r_1) \cup L(r_2))^R \) iff \( w \in L(r_1) \cup L(r_2) \) iff \( w \in L(r_1) \) or \( w \in L(r_2) \) iff \( w^R \in (L(r_1))^R \) or \( w^R \in (L(r_2))^R \) iff \( w^R \in L(r_1^R) \) or \( w^R \in L(r_2^R) \) (by the I.H.) iff \( w^R \in L(r_1^R + r_2^R) \) iff \( w^R \in L(r_1^R + r_2^R) = L(r^R) \) by item 4 in the procedure above.

5. \( r = r_1 r_2 \), for REs \( r_1 \) and \( r_2 \). Note that \( |r_1| < n \) and \( |r_2| < n \). Now \( w^R = u^R v^R \in (L(r))^R = (L(r_1) L(r_2))^R \) iff \( w = u v \in L(r_1) L(r_2) \) iff \( u \in L(r_1) \) and \( v \in L(r_2) \) iff \( u^R \in (L(r_1))^R \) and \( v^R \in (L(r_2))^R \) iff \( u^R \in L(r_1^R) \) and \( v^R \in L(r_2^R) \) (by the I.H.) iff \( w^R = u^R v^R \in L(r_1^R) L(r_2^R) \) iff \( w^R \in L(r_1^R + r_2^R) = L(r^R) \) by item 5 in the procedure above.

6. \( r = (r_1)^* \), for some RE \( r_1 \). Note that \( |r_1| < n \). Now \( w^R = w_1^R w_2^R \ldots w_n^R \in (L(r))^R = (L(r_1))^R \) iff \( w = w_1 w_2 \ldots w_n \in L(r_1)^R \) iff for all \( 1 \leq i \leq n \), \( w_i^R \in L(r_1^R) \) (by the I.H.) iff \( w^R = w_1^R w_2^R \ldots w_n^R \in L(r_1^R)^* = L(r^R) \) (by definition of \( * \)) by item 6 in the procedure above.

- Since in every case the procedure produces the correct output, the procedure is correct. Also note that the for any input \( r \) the procedure will terminate, since if \( |r| > 1 \) the procedure will be applied recursively to shorter REs, until eventually \( |r| = 1 \) at which point the procedure will terminate.

**Method 2.** Let \( r \) be a regular expression. Following the procedure given in Theorem 3.1 of Linz, we convert \( r \) to an nfa, \( N \), and then convert \( N \) to a dfa, \( D \). Following the procedure given in the solutions to 2.3.12 from HW1, convert \( D \) into an nfa, \( N^R \), such that \( L(N^R) = (L(D))^R \). Lastly, follow the procedure \( nfa \rightarrow rex \) given on page 83 of Linz, to convert \( L(N^R) \) into a RE, \( \hat{r} \). From the conversions it is clear, \( (L(r))^R = (L(N))^R = (L(D))^R = L(N^R) = L(\hat{r}) \).

3.2.18: Use the construction in Theorem 3.1 to find nfa’s for \( L(a \phi) \) and \( L(\phi^*) \). Is the result consistent with the definition of these languages.

By Theorem 3.1, the nfa for \( L(a) \) can be represented schematically as shown below:

```
q0 --a--> q1
```

The nfa for \( L(\emptyset) \) can represented as given below:

```
q\emptyset0 ----------- q\emptyset1
```

By Theorem 3.1, we can construct \( L(a \emptyset) \) using concatenation of the above two nfa’s as shown below:
The concatenated nfa can be further simplified as shown above. Since this nfa has no path to the final state $q_2$, the resulting language is $L(\emptyset)$. Also, we know that by definition, the concatenation of any language with $\emptyset$ results in $\emptyset$. So the resulting nfa is consistent with the definition of the language.

We already showed the construction of $L(\emptyset)$. By Theorem 3.1, we can construct $L(\emptyset^*)$ as shown below:

Since this nfa has a path to the final state $q_f$ on $\lambda$, the resulting language is $L(\lambda)$. Also, we know that by definition, the star of any regular expression will result in a language which has at least one element as $\lambda$ and by definition, $L(\emptyset^*)$ is $L(\lambda)$. So the resulting nfa is consistent with the definition of the language.

**Problem 2: Linz 3.3.6 and 3.3.14**

**3.3.6: Construct a right-linear grammar for the language $L((aab^*ab)^*)$.**

A right-linear grammar is as follows:

\[
S \rightarrow A|\lambda \\
A \rightarrow aaB \\
B \rightarrow bB|abS
\]
3.3.14: Show that for every regular language not containing $\lambda$ there exists a right linear grammar whose productions are restricted to the forms

$$A \rightarrow aB,$$

or

$$A \rightarrow a,$$

where $A, B \in V$, and $a \in T$.

In exercise problem 2.3.9, we proved that for any regular language $L$ that does not contain $\lambda$, there exists a nfa without $\lambda$-transitions and with a single final state that accept $L$. A new accept state $q_f$ was added and for every state $q_i$, if there is a transition from $q_i$ to a state in $F$ on input $a \in T$, a transition from $q_i$ to $q_f$ on input $a$ was added. Note that there were no outgoing transitions from this single final state, $q_f$. Also, every state in this nfa has a transition to some state in the nfa, except for the single final state, since we created this nfa by converting a dfa with extra transitions from $q_i$ to $q_f$ on input $a$ for every state $q_i$, if there was a transition from $q_i$ to a state in $F$ on input $a \in T$ in the original dfa.

Now we can modify the proof for Theorem 3.4 to construct the right linear grammar for a regular language not containing $\lambda$, while accommodating the restrictions to the forms of productions as mentioned above. Let $M = (Q, T, \delta, q_0, F)$ be a dfa that accepts a regular language $L$ not containing $\lambda$. Let the corresponding nfa as detailed in the above construction be $N = (Q \cup \{q_f\}, T, \delta', q_0, \{q_f\})$ where $q_f \notin Q$. Here $\delta'$ can be represented in terms of $\delta$ as:

$$\delta'(q_i, a) = \{q_k\} \text{ where } \delta(q_i, a) = q_k \text{ and } q_k \notin F,$$

$$\delta'(q_i, a) = \{q_k, q_f\} \text{ where } \delta(q_i, a) = q_k \text{ and } q_k \in F,$$

We assume that $Q = \{q_0, q_1, \ldots, q_n\}$ and $T = \{a_1, a_2, \ldots, a_m\}$. Construct the right-linear grammar $G = (V, T, S, P)$ with

$$V = \{q_0, q_1, \ldots, q_n\}$$

and $S = q_0$. Note that $q_f \notin V$. For each transition

$$\delta'(q_i, a_j) = \{q_k\} \text{ where } \delta(q_i, a_j) = q_k \text{ and } q_k \notin F,$$

of $N$, we put in $P$ the production

$$q_i \rightarrow a_j q_k$$

and for each transition

$$\delta'(q_i, a_j) = \{q_k, q_f\} \text{ where } \delta(q_i, a_j) = q_k \text{ and } q_k \in F,$$
of \( N \), we put in \( P \) the productions

\[
q_i \rightarrow a_j q_k \\
q_i \rightarrow a_j
\]

We first show that \( G \) defined in this way can generate every string in \( L \). Consider \( w \in L \) of the form

\[
w = a_i a_j \cdots a_k a_l.
\]

For \( N \) to accept this string, it must make moves via

\[
\delta'(q_0, a_i) = q_p, \\
\delta'(q_p, a_j) = q_r, \\
\vdots \\
\delta'(q_s, a_k) = q_t, \\
\delta'(q_t, a_l) = (q_u, q_f), q_u \in F.
\]

By construction, the grammar will have one production for each of these \( \delta' \)'s. Therefore, we can make the derivation

\[
q_0 \Rightarrow a_i q_p \Rightarrow a_i a_j q_r \Rightarrow^* a_i a_j \cdots a_k q_t \\
\Rightarrow a_i a_j \cdots a_k a_l,
\]

with the grammar \( G \), and \( w \in L(G) \). Conversely, if \( w \in L(G) \), then its derivation must have the form shown above. But this implies that

\[
\delta'^*(q_0, a_i a_j \cdots a_k a_l) = q_f
\]

Here note that the construction ensures the right linear grammar has productions restricted to the forms

\[
A \rightarrow aB, \\
A \rightarrow a,
\]

where \( A, B \in V \), and \( a \in T \).

**Problem 3: Linz 3.1.17**

3.1.17: Write regular expressions for the following languages on \( \{0, 1\} \).

(a): all strings ending in 01,

\[
(0 + 1)^*01
\]
(b): all strings not ending in 01,
\[(\lambda + 0 + 1 + (0 + 1)^* (00 + 10 + 11))\]

(c): all strings containing an even number of 0’s,
\[(1^*01^*01^*)^*\]

(d): all strings having at least two occurrences of the substring 00. (Note that with the usual interpretation of a substring, 000 contains two such occurrences),
\[(0 + 1)^* (00(0 + 1)^*00 + 000)(0 + 1)^*\]

(e): all strings with at most two occurrences of the substring 00,
\[(1 + 01)^* (001^*(01^*)^*00 + 000 + 00 + 0 + \lambda)(1 + 10)^*\]

(f): all strings not containing the substring 101.
\[0^*1^*0^* + (1 + 00 + 000)^* + (0^*1^*0^*)^*\]

Problem 4: Linz 3.2.5 and 3.2.9

3.2.5: Find dfa’s that accept the following languages.

(a) \(L = L(ab^*a^*) \cup L((ab)^*ba)\).

A dfa for \(L\) is given by:
(b) \( L = L(ab^*a^*) \cap L((ab)^*ba) \).

Since any string in \( L \) must start with an \( a \) to be in \( L_1 = L(ab^*a^*) \), it must also start with an \( ab \) or else it cannot be in \( L_2 = L((ab)^*ba) \). We cannot have a repeat of \( ab \) to be in \( L_1 \) (after the first \( b \), if we get an \( a \) we cannot get any more \( b \)'s). Therefore to be in \( L_2 \) the string must be \( abba \). This string is also in \( L_1 \). Thus, we see \( L = \{abba\} \). A DFA for \( L \) is then given by:

3.2.9: What language is accepted by the following generalized transition graph?

The regular expression corresponding to the language can be represented by:

\[ r = (a^*a(a + b)^*(a + b) + (a*b + c)(a + b)^*) \]

Problem 5: Linz 3.3.15

3.3.15: Show that any regular grammar \( G \) for which \( L(G) \neq \emptyset \) must have at least one production of the form

\[ A \rightarrow x \]

where \( A \in V \) and \( x \in T^* \).

For a contradiction, assume that \( G = (V, T, S, P) \) is regular with \( L(G) \neq \emptyset \), but with no productions of the form

\[ A \rightarrow x \]
That means all productions in $P$ are of the form

$$A \rightarrow xB$$

for some $A, B \in V$ and $x \in T^*$. Since $L(G) \neq \emptyset$, there must exist some $w \in T^*$ such that $w \in L(G)$. That means there is some derivation $S \Rightarrow^* w$ that ultimately terminates after producing $w$, which means there is some step in the derivation which does not produce a variable. However, this contradicts our assumption that all productions are of the form

$$A \rightarrow xB.$$

Thus, we conclude that at least one production in $P$ is of the form

$$A \rightarrow x.$$