Solutions of Problem Set 2 (Revised)

2.1 Given the grammar $G$:

\[
E \rightarrow E + T \mid T \\
T \rightarrow T \times F \mid F \\
F \rightarrow (E) \mid a
\]

give the parse trees and derivations for each string.

- **a.** The derivation is $E \Rightarrow T \Rightarrow F \Rightarrow a$. The parse tree is shown in Figure 1.
- **b.** The derivation is $E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T \Rightarrow a + F \Rightarrow a + a$. The parse tree is shown in Figure 1.
- **c.** The derivation is $E \Rightarrow E + T \Rightarrow E + T + T \Rightarrow T + T + T \Rightarrow F + T + T \Rightarrow a + T + T \Rightarrow a + F + T \Rightarrow a + a + T \Rightarrow a + a + F \Rightarrow a + a + a$. The parse tree is shown in Figure 1.
- **d.** The derivation is $E \Rightarrow T \Rightarrow F \Rightarrow (E) \Rightarrow (T) \Rightarrow (F) \Rightarrow ((E)) \Rightarrow ((T)) \Rightarrow ((F)) \Rightarrow ((a))$. The parse tree is shown in Figure 1.

![Figure 1: 2.1 (a), (b), (c), and (d)](image-url)

2.3 The given context-free grammar $G$ is
\[
\begin{align*}
R & \rightarrow XRX \mid S \\
S & \rightarrow aTb \mid bTa \\
T & \rightarrow XTX \mid X \mid \epsilon \\
X & \rightarrow a \mid b
\end{align*}
\]

a. The variables of \( G \) are \{ \( R, X, S, T \) \}. The terminals of \( G \) are \{ \( a, b \) \}. The start variable is \( R \).

b. Strings in \( L(G) \) are \( ab, ba \), and \( aaaabaaa \).

c. Strings not in \( L(G) \) are \( aaa, aba \), and \( bb \).

d. \( T \rightarrow aba \): false.

e. \( T \rightarrow^* aba \): true.

f. \( T \rightarrow T \): false.

g. \( T \rightarrow^* T \): true.

h. \( XXX \rightarrow^* aba \): true.

i. \( X \rightarrow^* aba \): false.

j. \( T \rightarrow^* XX \): true.

k. \( T \rightarrow^* XXX \): true.

l. \( S \rightarrow^* \epsilon \): false

m. The language generated by \( L \) is the language of all strings \( w \) over \{ \( a, b \) \} such that \( w \) is not palindrome, that is, \( w \neq w^R \).

2.6

b. \( L \) is the complement of the language \{ \( a^n b^n \mid n \geq 0 \) \}.

First, let’s see what the complement of \( L \) looks like:

\[
L = \{ a^n b^m \mid n \neq m \} \cup \{ (a \cup b)^* ba(a \cup b)^* \}
\]

Let’s call the leftmost language \( L_1 \) and the rightmost \( L_2 \).

The context-free grammar that generate \( L_1 \) is

\[
\begin{align*}
S_1 & \rightarrow aS_1b \mid T \mid U \\
T & \rightarrow aT \mid a \\
U & \rightarrow Ub \mid b
\end{align*}
\]

The context-free grammar that generate \( L_2 \) is

\[
\begin{align*}
S_2 & \rightarrow RbaR \\
R & \rightarrow RR \mid a \mid b \mid \epsilon
\end{align*}
\]

Therefore, the context-free grammar \( G \) that generate \( L = L_1 \cup L_2 \) is

\[
\begin{align*}
S & \rightarrow S_1 \mid S_2 \\
S_1 & \rightarrow aS_1b \mid T \mid U \\
S_2 & \rightarrow RbaR \\
T & \rightarrow aT \mid a \\
U & \rightarrow Ub \mid b \\
R & \rightarrow RR \mid a \mid b \mid \epsilon
\end{align*}
\]
c. \( L = \{ w\#x : w^R \text{ is a substring of } x \text{ for } w, x \in \{0,1\}^* \} \).

The context-free grammar \( G \) that generates \( L \) is

\[
S \rightarrow TR \\
T \rightarrow 0T0 | 1T1 | \#R \\
R \rightarrow RR | 0 | 1 | \epsilon
\]

2.7 a). The language of strings over the alphabet \( \{ a, b \} \) with twice as many \( a \)'s as \( b \)'s. The PDA that recognizes this language is shown in Figure 2.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2.png}
\caption{2.7 (a)}
\end{figure}

2.7 d). \( L = \{ x_1\#x_2 \cdots \#x_k | k \geq 1, \text{ each } x_i \in \{ a, b \} \text{ and for some } i \text{ and } j, x_i = x_j^R \} \).

The PDA of Figure 3 recognizes \( L \).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure3.png}
\caption{2.7 (d)}
\end{figure}
2.8 This sentence “the girl touches the boy with the flower” has two distinct parse trees as shown in Figure 4.
This shows that this CFG is ambiguous.

![Figure 4: Problem 2.8](image)

2.13 Given the grammar $G$:

$$
S \rightarrow TT \mid U \\
T \rightarrow 0T \mid T0 \mid \# \\
U \rightarrow 0U00 \mid \#
$$

a). The language generated by $L = L(G)$ is the set of strings that either are composed by the concatenation of 3 arbitrary-length strings of zeroes (delimited by the symbol \#) or strings of the form $0^k#0^{2k}$ for $k \geq 0$. More formally, a word $w$ in $L$ is of the form $w = 0^{k_1}#0^{k_2}#0^{k_3}$ (where $k_1, k_2, k_3 \geq 0$) or of the form $w = 0^k#0^{2k}$ for $k \geq 0$.

b). Let’s prove $L$ is not regular. Toward a contradiction, assume $L$ is regular. By the pumping lemma, there exists a pumping length $p > 0$ such that for any word $w \in L$, $|w| \geq p$, we can split word $w$ into 3 pieces $w = xyz$, $|y| > 0$, $|xy| \leq p$, such that for any integer $i \geq 0$, $w^i = xy^iz$ belongs to language $L$ too. Now, consider the word $w = 0^p#0^{2p}$. Clearly this word belongs to $L$ and $|w| \geq p$ so it must satisfy the conditions of the theorem. Any partition of word $w$ into 3 pieces $xyz$ much be such that $y$ is composed only by 0’s from the leftmost sequence of 0’s (otherwise $|xy|$ would be larger than $p$ which is not allowed). Therefore, $y = 0^k$ for $k > 0$. We now “pump down” this word, obtaining $w' = xy^0z$. Clearly, word $w' = 0^{p-k}#0^{2p}$ does not belong to $L$ because $k > 0$. So, any possible partition of $w$ cannot be pumped for all $i > 0$ (since $i = 0$ does not work!). We have obtained a contradiction. Therefore, $L$ must be non regular.
2.25 Notice that \( Y \) generates all possible strings. The grammar generates all strings NOT of the form \( a^k b^k \) for \( k \geq 0 \). Thus the complement of the language generated is \( \overline{L(G)} = \{ a^k b^k : k \geq 0 \} \).

The CFG for this language is very easy to generate:

\[
S \rightarrow aSb | \epsilon
\]

2.26 \( C = \{ x\#y : x, y \in \{0, 1\}^*, \text{ and } x \neq y \} \).

Solution: (Revised) The fundamental idea is that if \( x \) and \( y \) differ, there must be a position \( k \) in both words such that \( x_k \neq y_k \). Our PDA, while reading the first string, will count symbols (by pushing some other symbol \( x \) into the stack) until \textit{nondeterministically} will decide to store (remember) the current symbol in the stack. Then, when reading the second string it, the PDA will count symbols (by popping the \( x \)'s from the stack) until \textit{nondeterministically} will decide to check whether the current symbol matches the symbol stored in the stack. If they differ, it will accept. Otherwise it will reject.

![Figure 5: 2.26. PDA recognizing language \( C \).](image)

1.20 Give a formal definition of the finite state transducer (FST) model from exercise 1.19, following the patterns of Definition 1.1 (page 35 in the textbook). Assume that an FST has an input alphabet \( \Sigma \) and an output alphabet \( \Gamma \) but not a set of accepting states. Include a formal definition of the computation of an FST.

A finite state transducer (FST) is a 5-tuple \((Q, \Sigma, \Gamma, \delta, q_0)\) where

- \( Q \) is a finite set of states,
- \( \Sigma \) is a finite of symbols, called the input alphabet,
- \( \Gamma \) is a finite of symbols, called the output alphabet,
- \( \delta : Q \times \Sigma \rightarrow Q \times \Gamma_\epsilon \) is the transition function (where \( \Gamma_\epsilon = \Gamma \cup \{ \epsilon \} \)), and
- \( q_0 \) is the start state.

The computation of an FST proceeds as follows. Let \( M = (Q, \Sigma, \Gamma, \delta, q_0) \) be a FST, \( w = w_1, \ldots, w_n \) be a string over alphabet \( \Sigma \) and \( \sigma = \sigma_1, \ldots, \sigma_n \) be a string over alphabet \( \Gamma_\epsilon \) (ie. some of the \( \sigma_i \) can be \( \epsilon \)). The computation of \( M \) on input \( w \) produces output \( \sigma \) if there exists a sequence of states \( r_0, r_1, \ldots, r_n \) in \( Q \) such that the following conditions hold:

1. \( r_0 = q_0 \),
2. $\delta(r_{i-1}, w_i) = (r_i, \sigma_i)$, for $i = 1, \ldots, n$.

1.21 Using the solution you gave to Exercise 1.20, give a formal description of the machines $T_1$ and $T_2$ pictured in Exercise 1.19.

The description of FST $T_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, s_1)$ is as follows:

1. $Q_1 = \{ q_1, q_2 \}$
2. $\Sigma_1 = \{ 0, 1, 2 \}$
3. $\Gamma_1 = \{ 0, 1 \}$
4. $s_1 = q_1$
5. $\delta_1$ is given by the following table:

<table>
<thead>
<tr>
<th>$\delta_1$</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$q_1/0$</td>
<td>$q_1/1$</td>
<td>$q_2/1$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_1/0$</td>
<td>$q_2/1$</td>
<td>$q_2/1$</td>
</tr>
</tbody>
</table>

The description of FST $T_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2, s_2)$ is as follows:

1. $Q_2 = \{ q_1, q_2, q_3 \}$
2. $\Sigma_2 = \{ a, b \}$
3. $\Gamma_2 = \{ 0, 1 \}$
4. $s_2 = q_1$
5. $\delta_2$ is given by the following table:

<table>
<thead>
<tr>
<th>$\delta_2$</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$q_2/1$</td>
<td>$q_3/1$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_3/1$</td>
<td>$q_1/0$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_1/0$</td>
<td>$q_2/1$</td>
</tr>
</tbody>
</table>

1.32 Say that string $x$ is a prefix of a string $y$ if a string $z$ exists where $xz = y$ and that $x$ is a proper prefix of $y$ if in addition $x \neq y$. Show that the class of regular languages is closed under the following operation:

a) $\text{NOPREFIX}(A) = \{ w \in A : \text{no proper prefix of } w \text{ is a member of } A \}$.

Before doing anything, we need to be sure we understand the problem. To prove that a language $A$ is closed under a given operation (like $\text{NOPREFIX}$) we need to show that, given regular language $A$ we can show that the language $L = \text{NOPREFIX}(A)$ is also regular. How do we proceed? First, since the given language $A$ is regular, we know there exits a DFA (say $M$) that recognizes it. We will prove that $L = \text{NOPREFIX}(A)$ is regular by showing that there is a NFA (say $N$) that recognizes it. And how do we find $N$? We build $N$ by properly modifying DFA $M$ – we say we “construct” an NFA for $L$ using the DFA $M$. Once this is done, the final step is proving that the NFA we built is “correct”, namely that it recognizes language $L$ (formally, this means proving that if string $x$ is recognized by NFA
Then \( x \) belongs to \( L \) and, conversely, that if \( x \) belongs to \( L \) then it is recognized by \( N \). Details follow.

Assume there is a DFA \( M = (Q, \Sigma, \delta, q_0, F) \) that accepts \( A \). This means we are given all the components of \( N \), namely \( Q, \Sigma, \delta, q_0 \) and \( F \) – we can used them as we wish.

The language we want to build an NFA for is

\[
\text{NOPREFIX}(A) = \{ w \in A : \text{no proper prefix of } w \text{ is a member of } A \}
\]

An NFA for it doesn’t seem quite straightforward to build. So then? A strategy that usually works is to consider the complement of this language (or something close to the complement).

We consider the language formed by ALL the words \( w \) that do have a proper prefix in \( A \):

\[
L_1 = \{ w \in \Sigma^* : \text{some string } y \text{ in } A \text{ is a proper prefix of } w \} = \{ w \in \Sigma^* : \text{there is } y \in A \text{ and } v \in \Sigma^* \text{ s.t. } w = yv \}
\]

We construct an NFA \( N \) for this language. Building a NFA means showing all its components explicitly, meaning \( Q_1, \Sigma_1, \delta_1 \) and \( F_1 \) such that \( N = (Q_1, \Sigma_1, \delta_1, F_1) \). Here it is,

- \( Q_1 = Q \cup \{ q_f \} \), where \( q_f \not\in Q \) is a new state we have introduced,
- \( \Sigma_1 = \Sigma \) (no reason to change the input alphabet),
- \( \delta_1 \) is defined as follows. For any \( q \in Q_1 \) and any \( s \in \Sigma \cup \{ \epsilon \} \):
  \[
  \text{NOPREFIX}(A) = \begin{cases} 
  \{ \delta(q, s) \} & \text{if } q \in Q - F \text{ and } s \in \Sigma \\
  \{ \delta(q, s) \} \cup \{ q_f \} & \text{if } q \in F \text{ and } s \in \Sigma \\
  \{ q_f \} & \text{if } q = q_f \text{ and } s \in \Sigma \\
  \emptyset & \text{otherwise.}
  \end{cases}
  \]
- \( q_1 = q_0 \)
- \( F_1 = \{ q_f \} \)

We now prove the construction is correct.

- If \( w \) is a string in \( L_1 \), there is a string \( y \) in \( A \) which is a proper prefix of \( w \), or \( w = yx \), where \( x \) is not empty. If \( w \) is taken as the input of \( N \), some computation on \( y \) will end at an accepting state in \( M \), and then some computation on the \( x \) part will end at the state \( q_f \). This means \( w \) is accepted by \( N \).
- If \( w \) is a string accepted by \( N \). Then, there is some computation that ends at \( q_f \). By construction of \( N \), that computation must arrive at one of the accepting states in \( M \) before it ends at \( q_f \). If we call \( y \) to the proper prefix of \( w \) used in that part of the computation, we can see that \( M \) on input \( y \) will end up at one of its accepting states. This means \( y \) is in \( A \) and \( w \) is a member of \( L_1 \).

We conclude noticing that \( \text{NOPREFIX}(A) = A \cap \overline{L_1} \). Since the class of regular languages is closed under intersection and complement, we have that \( \text{NOPREFIX}(A) \) is regular too.