Problem 1

For each of the following languages, give a context-free grammar and a pushdown automaton. Give a short explanation of how the PDA works.

(a) $L_1 = \{wyw^R : \text{the length of } y \text{ is even}\}, \Sigma = \{a, b\}$. 
(Recall that $w^R$ is the reverse of $w$.)

(b) $L_2 = \{w : w \text{ has the same number of } a\text{s as } b\text{s and } c\text{s together}\}, \Sigma = \{a, b, c\}$.

(c) $L_3 = \{a^ib^jc^k : i > j \text{ or } j > k, \text{ where } i, j, k \geq 0\}, \Sigma = \{a, b, c\}$.

(d) (Extra credit) $L_4 = \{xy : |x| = |y| \text{ and } x \neq y\}, \Sigma = \{a, b\}$.

Solution

(a) This is a “trick question” as the language of $L_1$ is the set $(\Sigma\Sigma)^*$ of all strings with even length. So $L_1$ is described by the CFG $S \rightarrow abS \mid aaS \mid baS \mid bbS \mid \varepsilon$. You can also draw a 2-state PDA (based on the DFA of $L_1$) for $L_1$.

Alternatively, you can just follow the pattern in the definition of $L_1$ and come up with the following CFG and PDA.

$$
S \rightarrow aSa \mid bSb \mid Y \\
Y \rightarrow abY \mid bbY \mid baY \mid aaY \mid \varepsilon
$$
(b) To write a CFG, let $Y$ represent a $b$ or a $c$. If the string starts with an $a$, then this $a$ can be matched with an $X$ somewhere so that the remaining segments have the same number of $a$s and $X$s. Similarly, if it starts with an $Y$, this $Y$ can be matched with a later $a$. So we can generate the strings with the same number of $a$s and $Y$s via the rule

$$S \rightarrow aSYS \mid YSaS \mid \varepsilon$$

Finally, we add the rule $Y \rightarrow b \mid c$.

For the PDA, we use an $X$ on the stack to record the excess number of $a$s, and we use $Y$ to record the excess joint number of $b$s and $c$s. When we see an $a$, we have a choice: Either we can push an $X$ or we can pop a $Y$ (if it is available). When we see a $b$, we can either pop an $X$ or push a $Y$. If there are as many $b$s and $c$s together as there are $a$s, then the number of $Y$s on the stack will not exceed the number of $X$s. Moreover, there is always a way to run the PDA so there are always nothing left on the stack. On the other hand, if there are many $b$s and $c$s together as there are $a$s, it is not possible to have nothing left on the stack. So at the end, the PDA simply checks that nothing is left on the stack.
(c) There are two different possibilities: $i > j$ or $j > k$. $A,C$ generate the strings $a^*,c^*$ respectively. $X$ generates $a^ib^j$ where $i > j$, thus $XC$ represents the string $a^ib^jc^k$ where $i > j$. $Y$ generates $b^jc^k$, where $j > k$ thus $AY$ represents the string $a^ib^jc^k$, where $j > k$. Thus the union of $XC,AY$ represents $L_3$.

\[
S \rightarrow XC \mid AY \\
X \rightarrow aXb \mid aX \mid a \\
Y \rightarrow bYc \mid bY \mid b \\
A \rightarrow Aa \mid \varepsilon \\
C \rightarrow Cc \mid \varepsilon
\]

Similarly, to obtain a PDA, we draw two different PDAs for the two possibilities and connect them up via $\varepsilon$-transitions.
(d) Let’s first try to understand what the strings in $L_4$ look like. We know that $w \in L_4$ if and only if it can be written in the form $xy$ where $|x| = |y|$ but $x \neq y$. If $x$ and $y$ are different then they must differ in some position, say $i$. Then the string $w$ must look like

$$\begin{align*}
(a+b)^i a(a+b)^i (a+b)^i b(a+b)^j & \quad \text{or} \quad (a+b)^i b(a+b)^i (a+b)^j a(a+b)^j .
\end{align*}$$

We can rewrite each of these like

$$\begin{align*}
(a+b)^i a(a+b)^i (a+b)^i b(a+b)^j & \quad \text{or} \quad (a+b)^i b(a+b)^i (a+b)^j a(a+b)^j .
\end{align*}$$

So we can write language $L_5$ as $L_a L_b \cup L_b L_a$, where $L_a = \{ uaw : |u| = |w| \}$ and $L_b$ is defined similarly. Clearly $L_a$ and $L_b$ are context-free as they are described by the CFGs:

$$\begin{align*}
S_a & \rightarrow US_a U \mid a \\
S_b & \rightarrow US_b U \mid b \\
U & \rightarrow a \mid b
\end{align*}$$

To obtain a CFG for $L_4$, we add the starting production:

$$S \rightarrow S_a S_b \mid S_b S_a.$$
We can design a PDA following a similar principle. First, we design a PDA for \( L_a \). The PDA pushes \( x \)'s on the stack for each input symbol until it non-deterministically the middle \( a \), then it pops \( x \)'s for the remaining symbols until the end of the stack is reached, and accepts if the end of the stack coincides with the end of the input. The PDA for \( L_b \) is very similar. To obtain PDAs for \( L_a L_b \) and \( L_b L_a \), we combine the two PDAs back-to-back. Finally, the PDA for \( L_5 \) is obtained by combining these two via \( \varepsilon \)-transitions.

Here is a somewhat simplified version of the PDA obtained by this construction.

Problem 2

Consider the following context-free grammar \( G \) that describes (nontrivial) regular expressions over the alphabet \( \{0, 1\} \):

\[
R \rightarrow (R) | R+R | RR | R* | 0 | 1 | e
\]

The alphabet of \( G \) consists of the symbols (, ), +, *, 0, 1, and \( e \). Here + and * describe the union and star operators, while \( e \) describes the empty string.

(a) Convert \( G \) to Chomsky Normal Form.

(b) Apply the Cocke-Younger-Kasami algorithm (algorithm 2 from lecture 9) to obtain parse trees for the following strings: \((1+0)*\), \(0+01\), \((1+e)1*\). Some of these expressions several parse trees; which ones describe the intended meaning of the expression?

(c) Give a CFG \( G' \) that describes the same language as \( G \) but is not ambiguous. Moreover, each parse tree in \( G' \) should describe the intended meaning of the corresponding regular expression.
Solution

(a) We notice that there are no unit productions and $\varepsilon$-productions in original CFG. So we just break up long sequences with new variables. We obtain the following grammar:

\[
\begin{align*}
R &\rightarrow XB \mid YR \mid RR \mid RM \mid 0 \mid 1 \mid \varepsilon \\
X &\rightarrow AR \\
Y &\rightarrow RP \\
A &\rightarrow ( \\
B &\rightarrow ) \\
P &\rightarrow + \\
M &\rightarrow *
\end{align*}
\]

(b) On input $1+0\ast$, the run of the CYK algorithm looks like this:

\[
\begin{array}{c c c c c}
& R & R & X & Y & A R P R B M \\
\hline
R & & & & 1 & + & 0 & * \\
X & & & R & & & & \\
\hline
& & & & & & & & \\
& & & & & & & & \\
& & & & & & & & \\
& & & & & & & ( & 1 & + & 0 & ) & & & & * \\
\end{array}
\]

The table yields the following unique parse tree:

On input $0+01$,
There are two parse trees for this expression. Only the left one describes the intended meaning.

(c) To disambiguate the grammar and ensure the parse tree has the intended meaning, we have to consider the order of precedence of the operators (union, concatenation, and star). Union has the least precedence, so we write a regular expression $R$ as a union of one or more terms (which we represent by $T$). Next down the line is concatenation, so each term is a concatenation of
one or more factors (F). Now each factor can be the star of another factor, or it can represent a whole regular expression, provided it is parenthesized. Finally, the “atomic” factors are the constants 0, 1, and e, which we represent by C.

\[
\begin{align*}
R & \rightarrow R + T \mid T \\
T & \rightarrow TF \mid F \\
F & \rightarrow (R) \mid F^* \mid C \\
C & \rightarrow 0 \mid 1 \mid e
\end{align*}
\]

Problem 3

Consider the following languages. For each of the languages, say whether the language is (1) regular, (2) context-free but not regular, or (3) not context free. Explain your answer (e.g. give a DFA or argue why one exists, give a CFG, apply the appropriate pumping lemma).

(a) \(L_1 = \{a^n b^n a^n b^n : n \geq 0\}, \Sigma = \{a, b\}\).

(b) \(L_2 = \{w^R \# z : w \text{ is a substring of } z, w, z \in \{a, b\}^*\}, \Sigma = \{a, b, \#\}\).

(c) \(L_3 = \{w \# z : w \text{ is a substring of } z, w, z \in \{a, b\}^*\}, \Sigma = \{a, b, \#\}\).

(d) \(L_4 = \{x + y = z : x + y = z \text{ in unary where } x, y, z \in 11^*\}, \Sigma = \{1, =, +\}\). For example, 1+11 = 111 \(\in L_4\) but +1 = 1 \(\notin L_4\), 1 + 1 = 111 \(\notin L_4\).

Solution

(a) \(L_1\) is not context-free. Suppose it is. By the pumping lemma, there is a pumping length \(p\) so that every \(w \in L_1\) of length at least \(p\) can be written as \(w = uvxyz\) so that \(|vy| > 0\), \(|vxy| \leq p\) and \(uv^ixy^iz \in L_1\) for every \(i\). Let \(w = a^n b^n a^n b^n\), which is in \(L_1\) and consider any way of writing \(w\) as \(uvxyz\). We consider three cases:

**Case 1:** \(vy\) is of the form \(a^*\) or \(b^*\). Then \(vxy\) must be contained in a contiguous block of \(a\)s or a contiguous block of \(b\)s. Suppose it is a block of \(a\)s. Then \(uv^2xy^2z\) contains an uneven number of \(a\)s in the two blocks, so it is not in \(L_1\). If it is a block of \(b\)s, the analysis is analogous.

**Case 2:** \(vy\) contains both \(a\)s and \(b\)s. Then \(uv^2xy^2z\) is not of the form \(a^*b^*a^*b^*\), so it is not in \(L_1\).

This contradicts the assumption that \(L_1\) is context-free.

(b) \(L_2\) is context free but not regular. We first give a CFG. The strings in \(L_1\) have the form \(w^R \# uwz\), where \(u, w, z \in \{a, b\}^*\). A string \(S\) of this form can be written as \(Az\), where
\( z \in \{a, b\}^* \) and \( A \) has the form \( w^R \# uw \). We can write \( A \) as \( w^R B w \), where \( B \) has the form \( \#(a+b)^n \) and \( w \in \{a, b\}^* \). This gives the following CFG:

\[
S \to Sa \mid Sb \mid A \\
A \to aAa \mid bAb \mid B \\
B \to Ba \mid Bb \mid \#
\]

To show \( L_2 \) is not regular, for pumping length \( n \), we choose \( s = a^n \# a^n \) which is in \( L_2 \). For any partition \( s = xyz \) where \( y \neq \varepsilon \) and \( |xy| \leq n \), \( y \) must contain only 1s before +. Suppose \( y = a^j \), where \( 0 < j \leq n \). Then we can let \( i = 2 \) and get \( xy^jz = xy^2z = a^{n-j}a^j \# a^n = a^{n+j} \# a^n \) which is not in \( L_2 \). Hence \( L_2 \) is not regular.

(c) \( L_3 \) is not context-free. We prove this by pumping lemma. Consider an arbitrary pumping length \( n \), and choose \( s = a^n b^n \# a^n b^n \). Then \( s \in L_3 \) but we will show that no matter how we write \( s = uvwxy \) with \(|vwx| \leq n, |x| > 0 \), we can pump it out of \( L_3 \). We look at three cases:

- Case 1: \( vwx \) is in the first half (before \( \# \)) of \( a^n b^n \# a^n b^n \). We choose \( i = 2 \), thus \( uv^2w^2x^2y \) looks like \( z \# z' \) where \(|z| > |z'| \) so that \( z \) cannot be a substring of \( z' \). Hence \( uv^2w^2x^2y \) is not in \( L_3 \).

- Case 2: \( vwx \) is in the second half (after \( \# \)) of \( a^n b^n \# a^n b^n \). We choose \( i = 0 \), thus \( uv^0w^0x^0y \) looks like \( a^n b^n \# a^n b^j \) where \( i < n \) or \( j < n \) (or both). \( a^n b^n \) can not be a substring of \( a^n b^j \).

- Case 3: \( vwx \) is in the middle part of \( a^n b^n \# a^n b^n \) (not intersecting the initial block of \( a^n \) and the final block of \( b^n \)). If \( vwx \) contains \( \# \), then we choose \( i = 0 \), thus \( uv^0w^0x^0y \) contains no \( \# \) so that \( uv^0w^0x^0y \) is not in \( L_3 \). Else, \( v \) must in left bs of \( \# \) and \( x \) in the right as of \( \# \). If \( v \neq \varepsilon \), we choose \( i = 2 \), then there are more bs in the left string of \( \# \) than the right string of \( \# \) in \( uv^2w^2x^2y \). If \( x \neq \varepsilon \), we choose \( i = 0 \), then there are more as in the left string of \( \# \) than the right string of \( \# \) in \( uv^0w^0x^0y \). In those cases, \( uv^iwx^iy \) is out of \( L_3 \) since the left string of \( \# \) can be a substring of right string of \( \# \).

This covers all the cases, so by the pumping lemma for context-free languages, \( L_3 \) is not context-free.

(d) \( L_4 \) is context-free but not regular. We can give following grammar for this language

\[
S \to A + B = AB \\
A \to 1A \mid 1 \\
B \to 1B \mid 1
\]

We prove it’s not regular by pumping lemma. Consider an arbitrary pumping length \( n \), and choose \( s = 1^n + 1^n = 1^{2n} \). For any partition \( s = xyz \) where \( y \neq \varepsilon \) and \( |xy| \leq n \), \( y \) must contain only 1s before +. Suppose \( y = 1^j \), where \( 0 < j \leq n \). Then we can let \( i = 2 \) and get \( xy^jz = xy^2z = 1^{n-j}1^j + 1^n = 1^{2n} \) which is not in \( L_2 \). Hence \( L_2 \) is not regular.
Problem 4

Context-free grammars are sometimes used to model natural languages. In this problem you will model a fragment of the English language using context-free grammars. Consider the following English sentences:

The girl is pretty.
The girl that the boy likes is pretty.
The girl that the boy that the clerk pushed likes is pretty.
The girl that the boy that the clerk that the girl knows pushed likes is pretty.

This is a special type of sentence built from a subject (The girl), a relative pronoun (that) followed by another sentence, a verb (is) and an adjective (pretty).

(a) Give a context-free grammar $G$ that models this special type of sentence. Your terminals should be words or sequences of words like pretty or the girl.

(b) Is the language of $G$ regular? If so, write a regular expression for it. If not, prove using the pumping lemma for regular languages.

(c) Can you give an example of a sentence that is in $G$ but does not make sense in common English?

Solution

(a) The following grammar $G$ describes this type of sentence:

\[
\begin{align*}
\text{SENTENCE} & \rightarrow \text{SUBJ REL VERB ADJ} \\
\text{REL} & \rightarrow \text{that SUBJ REL VERB} | \epsilon \\
\text{SUBJ} & \rightarrow \text{the girl} | \text{the boy} | \text{the clerk} \\
\text{VERB} & \rightarrow \text{is} | \text{likes} | \text{pushed} | \text{knows} \\
\text{ADJ} & \rightarrow \text{pretty}
\end{align*}
\]

(b) This language is not regular. We prove this via the pumping lemma for regular languages. For any given $n$, the string

\[
\text{the girl (that the boy)}^n \ (\text{knows})^n \ \text{is pretty}
\]

is in $L(G)$. It can be generated via the recursive rule for $REL$. Let $uvw$ be any splitting of $z$ where $|uv| \leq n$, $|v| > 0$. If $v$ consists only of a single copy of “that”, then $uv^2w$ is not in $L(G)$ since it contains the pattern “that that”, which is not allowed by the rules. Otherwise $v$ contains at least one subject (“the girl” or “the boy”) but no verb. Then $uv^2w$ has more subjects than verbs, which is not allowed by the rules of the grammar, where every subject must have a matching verb.
(c) You can give different kinds of examples. For instance, “The girl pushed pretty” does not make sense in English. Neither does “The boy that the boy is is pretty”. There are also examples that make grammatical sense, but we would hardly use in practice, like

The girl that the boy that the clerk that the girl knows pushed likes is pretty.

This illustrates some of the difficulties in trying to design formal grammars for natural languages.