Part III: Context-Free Languages and Pushdown Automata

1 Context-Free Grammars

6) Show a context-free grammar for each of the following languages L:

a) BalDelim = $\{w$: where w is a string of delimeters: (,), [,], {, }, that are properly balanced$\}$.

$S \rightarrow (S) \mid [S] \mid \{ S \} \mid S S \mid \varepsilon$

b) $\{\mathtt{a}^i\mathtt{b}^j : 2i = 3j + 1\}$.

$S \rightarrow \mathtt{aaa}S\mathtt{bb} \mid \mathtt{aab}$

c) $\{\mathtt{a}^i\mathtt{b}^j : 2i \neq 3j + 1\}$.

We can begin by analyzing L, as shown in the following table:

# of a's	Allowed # of b's
0	any
1	any
2	any except 1
3	any
4	any
5	any except 3
6	any
7	any
8	any except 5

$S \rightarrow \mathtt{aaa}S\mathtt{bb}$
$S \rightarrow \mathtt{aaa}X$ /* extra a's
$S \rightarrow T$ /* terminate
$X \rightarrow A \mid A\ \mathtt{b}$ /* arbitrarily more a's
$T \rightarrow A \mid B \mid \mathtt{a}\ B \mid \mathtt{aabb}\ B$ /* note that if we add two more a's we cannot add just a single b.
$A \rightarrow \mathtt{a}\ A \mid \varepsilon$
$B \rightarrow \mathtt{b}\ B \mid \varepsilon$

d) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* : \#_{\mathtt{a}}(w) = 2\ \#_{\mathtt{b}}(w)\}$.

$S \rightarrow S\mathtt{a}S\mathtt{a}S\mathtt{b}S$
$S \rightarrow S\mathtt{a}S\mathtt{b}S\mathtt{a}S$
$S \rightarrow S\mathtt{b}S\mathtt{a}S\mathtt{a}S$
$S \rightarrow \varepsilon$

e) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* : w = w^R\}$.

$S \rightarrow \mathtt{a}S\mathtt{a}$
$S \rightarrow \mathtt{b}S\mathtt{b}$
$S \rightarrow \varepsilon$
$S \rightarrow \mathtt{a}$
$S \rightarrow \mathtt{b}$ }

f) $\{a^i b^j c^k : i, j, k \geq 0 \text{ and } (i \neq j \text{ or } j \neq k)\}$.

$S \rightarrow XC$ /* $i \neq j$
$S \rightarrow AY$ /* $j \neq k$
$X \rightarrow aXb$
$X \rightarrow A'$ /* at least one extra a
$X \rightarrow B'$ /* at least one extra b
$Y \rightarrow bYc \mid B' \mid C'$
$A' \rightarrow a\ A' \mid a$
$B' \rightarrow b\ B' \mid b$
$C' \rightarrow c\ C' \mid c$
$A \rightarrow a\ A \mid \varepsilon$
$C \rightarrow c\ C \mid \varepsilon$ }

g) $\{a^i b^j c^k : i, j, k \geq 0 \text{ and } (k \leq i \text{ or } k \leq j)\}$.

$S \rightarrow A \mid B$
$A \rightarrow aAc \mid aA \mid M$
$B \rightarrow aB \mid F$
$F \rightarrow bFc \mid bF \mid \varepsilon$
$M \rightarrow bM \mid \varepsilon$

h) $\{w \in \{a, b\}^* : \text{every prefix of } w \text{ has at least as many a's as b's}\}$.

$S \rightarrow aS \mid aSb \mid SS \mid \varepsilon$

i) $\{a^n b^m : m \geq n, m\text{-}n \text{ is even}\}$.

$S \rightarrow aSb \mid S \rightarrow Sbb \mid \varepsilon$

j) $\{a^m b^n c^p d^q : m, n, p, q \geq 0 \text{ and } m + n = p + q\}$.

For any string $a^m b^n c^p d^q \in L$, we will produce a's and d's in parallel for a while. But then one of two things will happen. Either $m \geq q$, in which case we begin producing a's and c's for a while, or $m \leq q$, in which case we begin producing b's and d's for a while. (You will see that it is fine that it is ambiguous what happens if $m = q$.) Eventually this process will stop and we will begin producing the innermost b's and c's for a while. Notice that any of those four phases could produce zero pairs. Since the four phases are distinct, we will need four nonterminals (since, for example, once we start producing c's, we do not want ever to produce any d's again). So we have:

$S \rightarrow aSd$
$S \rightarrow T$
$S \rightarrow U$
$T \rightarrow aTc$
$T \rightarrow V$
$U \rightarrow bUd$
$U \rightarrow V$
$V \rightarrow bVc$
$V \rightarrow \varepsilon$

k) $\{x\mathtt{c}^n : x \in \{\mathtt{a}, \mathtt{b}\}^*$ and $(\#_{\mathtt{a}}(x) = n$ or $\#_{\mathtt{b}}(x) = n)\}$.

$S \rightarrow A \mid B$
$A \rightarrow B'\ \mathtt{a}\ B'\ A\ \mathtt{c} \mid B'$
$B' \rightarrow \mathtt{b}B' \mid \varepsilon$
$B \rightarrow A'\ \mathtt{b}\ A'\ B\ \mathtt{c} \mid A'$
$A' \rightarrow \mathtt{a}A' \mid \varepsilon$

l) $\{b_i \# b_{i+1}{}^{\mathrm{R}} : b_i$ is the binary representation of some integer i, $i \geq 0$, without leading zeros$\}$. (For example $\mathtt{101\#011} \in L$.)

L can be written as:
$\mathtt{0\#1} \cup \{\mathtt{1}^k\mathtt{\#0}^k\mathtt{1} : k > 0\} \cup \{u\mathtt{01}^k\mathtt{\#0}^k\mathtt{1}u^{\mathrm{R}} : k \geq 0$ and $u \in \mathtt{1}(\mathtt{0} \cup \mathtt{1})^*\}$
So a grammar for L is:
$S \rightarrow \mathtt{0\#1} \mid \mathtt{1}\ S_1\ \mathtt{1} \mid \mathtt{1}\ S_2\ \mathtt{1}$
$S_1 \rightarrow \mathtt{1}\ S_1\ \mathtt{0} \mid \mathtt{\#0}$
$S_2 \rightarrow \mathtt{1}\ S_2\ \mathtt{1} \mid \mathtt{0}\ S_2\ \mathtt{0} \mid \mathtt{0}\ A\ \mathtt{1}$
$A \rightarrow \mathtt{1}\ A\ \mathtt{0} \mid \mathtt{\#}$

m) $\{x^{\mathrm{R}}\#y : x, y \in \{\mathtt{0}, \mathtt{1}\}^*$ and x is a substring of $y\}$.

$S \rightarrow S\mathtt{0} \mid S\mathtt{1} \mid S_1$
$S_1 \rightarrow \mathtt{0}S_1\mathtt{0} \mid \mathtt{1}S_1\mathtt{1} \mid T$
$T \rightarrow T\mathtt{1} \mid T\mathtt{0} \mid \mathtt{\#}$

8) Consider the unambiguous expression grammar G' of Example 11.19.
   a. Trace a derivation of the string `id+id*id*id` in G'.

$E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow \mathtt{id} + T \Rightarrow \mathtt{id} + T * F \Rightarrow \mathtt{id} + T * F * F \Rightarrow \mathtt{id} + F * F * F \Rightarrow$
$\mathtt{id} + \mathtt{id} * F * F \Rightarrow \mathtt{id} + \mathtt{id} * \mathtt{id} * F \Rightarrow \mathtt{id} + \mathtt{id} * \mathtt{id} * \mathtt{id}$

   b. Add exponentiation (**) and unary minus (-) to G', assigning the highest precedence to unary minus, followed by exponentiation, multiplication, and addition, in that order.

$R = \{ E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow F ** X$
$F \rightarrow X$
$X \rightarrow -X$
$X \rightarrow Y$
$Y \rightarrow (E)$
$Y \rightarrow \mathtt{id}\ \}$.

17) Consider the grammar G' of Example 11.19.

a) Convert G' to Chomsky normal form.

Remove-units produces:

$E \rightarrow E + T \mid T * F \mid (E) \mid \texttt{id}$
$T \rightarrow T * F \mid (E) \mid \texttt{id}$
$F \rightarrow (E) \mid \texttt{id}$

Then the final result is:

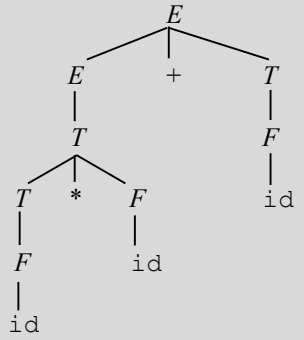
$E \rightarrow E\ E_1 \mid T\ E_2 \mid T_(\ E_3 \mid \texttt{id}$
$T \rightarrow T\ E_2 \mid T_(\ E_3 \mid \texttt{id}$
$F \rightarrow T_(\ E_3 \mid \texttt{id}$
$E_1 \rightarrow T_+\ T$
$E_2 \rightarrow T_*\ F$
$E_3 \rightarrow E\ T_)$
$T_(\rightarrow (\ )$
$T_) \rightarrow)$
$T_+ \rightarrow +$
$T_* \rightarrow *$

b) Consider the string `id*id+id`.

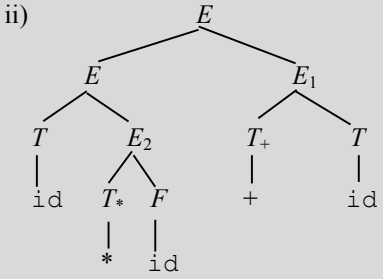
i) Show the parse tree that G' produces for it.

ii) Show the parse tree that your Chomsky normal form grammar produces for it.

i)



ii)



18) Convert each of the following grammars to Chomsky Normal Form:

a) $S \rightarrow \texttt{a}\ S\ \texttt{a}$
$S \rightarrow B$
$B \rightarrow \texttt{b}\ \texttt{b}\ C$
$B \rightarrow \texttt{b}\ \texttt{b}$
$C \rightarrow \varepsilon$
$C \rightarrow \texttt{c}\ C$

Step 1: get rid of ε productions:

Nullable variables: C

$S \to \mathtt{a}\ S\ \mathtt{a}$
$S \to B$
$B \to \mathtt{b}\ \mathtt{b}\ C$
$B \to \mathtt{b}\ \mathtt{b}$
$C \to \mathtt{c}\ C$
$C \to \mathtt{c}$

Step 2: eliminate unit productions:

Remove $S \to B$:

$S \to \mathtt{a}\ S\ \mathtt{a}$
$S \to \mathtt{b}\ \mathtt{b}\ C$
$S \to \mathtt{b}\ \mathtt{b}$
$B \to \mathtt{b}\ \mathtt{b}\ C$
$B \to \mathtt{b}\ \mathtt{b}$
$C \to \mathtt{c}\ C$
$C \to \mathtt{c}$

Step 3: eliminate terminals whenever the length of the right hand side is greater than 1:

$S \to T_a\ S\ T_a$
$S \to T_b\ T_b\ C$
$S \to T_b\ T_b$
$B \to T_b\ T_b\ C$
$B \to T_b\ T_b$
$C \to T_c\ C$
$C \to \mathtt{c}$
$T_a \to \mathtt{a}$
$T_b \to \mathtt{b}$
$T_c \to \mathtt{c}$

Step 4: Eliminate long rules:

$S \to T_a\ S'$
$S' \to S\ T_a$
$S \to T_b\ S''$
$S'' \to T_b\ C$
$S \to T_b\ T_b$
$B \to T_b\ B'$
$B' \to T_b\ C$
$B \to T_b\ T_b$
$C \to T_c\ C$
$C \to \mathtt{c}$
$T_a \to \mathtt{a}$
$T_b \to \mathtt{b}$
$T_c \to \mathtt{c}$

Note that our algorithm retains the rules with B on the left hand side. But they will never be used in any derivation because they are unreachable from S. If you want, you can add a simplification step that removes unreachable rules, which would yield:

$S \to T_a\ S'$
$S' \to S\ T_a$
$S \to T_b\ S''$
$S'' \to T_b\ C$
$S \to T_b\ T_b$
$C \to T_c\ C$
$C \to \mathtt{c}$
$T_a \to \mathtt{a}$
$T_b \to \mathtt{b}$
$T_c \to \mathtt{c}$

b) $S \to ABC$
$A \to \mathtt{a}C \mid D$
$B \to \mathtt{b}B \mid \varepsilon \mid A$
$C \to A\mathtt{c} \mid \varepsilon \mid C\mathtt{c}$
$D \to \mathtt{aa}$

$S \rightarrow AS_1$
$S_1 \rightarrow BC$
$S \rightarrow AC$
$S \rightarrow AB$
$S \rightarrow X_aC \mid \mathtt{a} \mid X_aX_a$
$A \rightarrow X_aC$
$A \rightarrow \mathtt{a}$
$A \rightarrow X_aX_a$
$B \rightarrow X_bB$
$B \rightarrow \mathtt{b}$
$B \rightarrow \varepsilon$
$B \rightarrow X_aC \mid \mathtt{a} \mid X_aX_a$

$C \rightarrow AX_c$
$C \rightarrow \varepsilon$
$C \rightarrow CX_c$
$C \rightarrow \mathtt{c}$
$D \rightarrow X_aX_a$
$X_a \rightarrow \mathtt{a}$
$X_b \rightarrow \mathtt{b}$
$X_c \rightarrow \mathtt{c}$

c) $S \rightarrow \mathtt{a}TV\mathtt{a}$
$T \rightarrow \mathtt{a}T\mathtt{a} \mid \mathtt{b}T\mathtt{b} \mid \varepsilon \mid V$
$V \rightarrow \mathtt{c}V\mathtt{c} \mid \varepsilon$

$S \rightarrow AS_1 \mid AS_2 \mid AS_3 \mid AA$
$S_1 \rightarrow TS_2$
$S_2 \rightarrow VA$
$S_3 \rightarrow TA$
$T \rightarrow AA \mid BB \mid CC \mid CT_1 \mid AS_3 \mid BT_2$
$T_2 \rightarrow TB$

$V \rightarrow CT_1 \mid CC$
$T_1 \rightarrow VC$
$A \rightarrow \mathtt{a}$
$B \rightarrow \mathtt{b}$
$C \rightarrow \mathtt{c}$

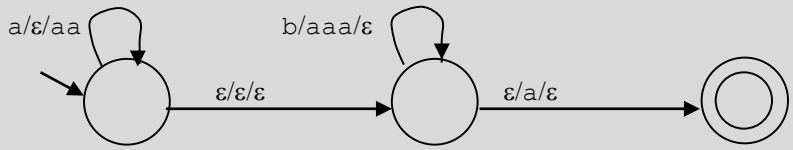
2 Pushdown Automata

1) Build a PDA to accept each of the following languages L:

a) BalDelim = {w : where w is a string of delimeters: (,), [,], {, }, that are properly balanced}.

$M = (\{1\}, \{(,), [,], \{, \}\}, \{(, [, \{\}, \Delta, 1, \{1\})$, where $\Delta =$
{ ((1, (, ε), (1, ())),
((1, [, ε), (1, [])),
((1, {, ε), (1, {})),
((1,),)), (1, ε)),
((1,],]), (1, ε)),
((1, }, }), (1, ε)) }

b) $\{a^i b^j : 2i = 3j + 1\}$.



c) $\{w \in \{a, b\}^* : \#_a(w) = 2 \cdot \#_b(w)\}$.

The idea is that we only need one state. The stack will do all the work. It will count whatever it is ahead on. Since one a matches two b's, each a will push an a (if the machine is counting a's) and each b (if the machine is counting a's) will pop two of them. If, on the other hand, the machine is counting b's, each b will push two b's and each a will pop one. The only tricky case arises with inputs like aba. M will start out counting a's and so it will push one onto the stack. Then comes a b. It wants to pop two a's, but there's only one. So it will pop that one and then switch to counting b's by pushing a single b. The final a will then pop that b. M is highly nondeterministic. But there will be an accepting path iff the input string w is in L.

$M = (\{1\}, \{a, b\}, \{a, b\}, \Delta, 1, \{1\})$, where $\Delta =$
{ ((1, a, ε), (1, a)),
((1, a, b), (1, ε)),
((1, b, ε), (1, bb)),
((1, b, aa), (1, ε)),
((1, b, a), (1, b)) }

d) $\{a^n b^m : m \le n \le 2m\}$.

$M = (\{1, 2\}, \{a, b\}, \{a\}, \Delta, 1, \{1, 2\})$, where $\Delta =$
{ ((1, a, ε), (1, a)),
((1, ε, ε), (2, ε)),
((2, b, a), (2, ε)),
((2, b, aa), (2, ε)) }.

e) $\{w \in \{a, b\}^* : w = w^R\}$.

This language includes all the even-length palindromes of Example 12.5, plus the odd-length palindromes. So a PDA to accept it has a start state we'll call 1. There is a transition, from 1, labeled ε/ε/ε, to a copy of

the PDA of Example 12.5. There is also a similarly labeled transition from 1 to a machine that is identical to the machine of Example 12.5 except that the transition from state *s* to state *f* has the following two labels: a/ε/ε and b/ε/ε. If an input string has a middle character, that character will drive the new machine through that transition.

3 Context-Free and Noncontext-Free Languages

1) For each of the following languages L, state whether L is regular, context-free but not regular, or not context-free and prove your answer.

a) $\{xy : x, y \in \{a, b\}^* \text{ and } |x| = |y|\}$.

Regular. $L = \{w \in \{a, b\}^* : |w| \text{ is even}\}$
$= (aa \cup ab \cup ba \cup bb)^*$

b) $\{(ab)^n a^n b^n : n > 0\}$.

Not context-free. We prove this with the Pumping Theorem. Let $w = (ab)^k a^k b^k$. Divide w into three regions: the ab region, the a region, and the b region. If either v or y crosses the boundary between regions 2 and 3 then pump in once. The resulting string will have characters out of order. We consider the remaining alternatives for where nonempty v and y can occur:
(1, 1) If $|vy|$ is odd, pump in once and the resulting string will have characters out of order. If it is even, pump in once. The number of ab's will no longer match the number of a's in region 2 or b's in region 3.
(2, 2) Pump in once. More a's in region 2 than b's in region 3.
(3, 3) Pump in once. More b's in region 3 than a's in region 2.
v or y crosses the boundary between 1 and 2: Pump in once. Even if v and y are arranged such that the characters are not out of order, there will be more ab pairs than there are b's in region 3.
(1, 3) $|vxy|$ must be less than or equal to k.

c) $\{x\#y : x, y \in \{0, 1\}^* \text{ and } x \neq y\}$.

Context-free not regular. We can build a PDA M to accept L. All M has to do is to find one way in which x and y differ. We sketch its construction: M starts by pushing a bottom of stack marker Z onto the stack. Then it nondeterministically chooses to go to state 1 or 2. From state 1, it pushes the characters of x, then starts popping the characters of y. It accepts if the two strings are of different lengths. From state 2, it must accept if two equal length strings have at least one different character. So M starts pushing a % for each character it sees. It nondeterministically chooses a character on which to stop. It remembers that character in its state (so it branches and there are two similar branches from here on). It reads the characters up to the # and does nothing with them. Starting with the first character after the #, it pops one % for each character it reads. When the stack is empty it checks to see whether the next input character matches the remembered character. If it does not, it accepts.

d) $\{a^i b^n : i, n > 0 \text{ and } i = n \text{ or } i = 2n\}$.

Context-free, not regular. L can be generated by the following context-free grammar:

$S \rightarrow A \mid B$
$A \rightarrow aAb \mid ab$
$B \rightarrow aaBb \mid aab$

L is not regular, which we show by pumping: Let $w = a^k b^k$. Pump out. To get a string in L, i must be equal to n or greater than n (in the case where $i = 2n$). Since we started with $i = n$ and then decreased i, the resulting string must not be in L.

e) $\{wx : |w| = 2 \cdot |x| \text{ and } w \in a^+ b^+ \text{ and } x \in a^+ b^+\}$.

Context-free, not regular. L can be accepted by a PDA M that pushes one character for each a and b in w and then pops two characters for each a and b in x.

L is not regular, which we show by pumping. Note that the boundary between the w region and the x region is fixed; it's immediately after the last b in the first group. Choose to pump the string $w = a^{2k} b^{2k} a^k b^k$. $y =$

a^p, for some nonzero p. Pump in or out. The length of w changes but the length of x does not. So the resulting string is not in L.

f) $\{a^n b^m c^k: n, m, k \geq 0 \text{ and } m \leq min(n, k)\}$.

Not context-free. We prove it using the Pumping Theorem. Let k be the constant from the Pumping Theorem and let $w = a^k b^k c^k$. Let region 1 contain all the a's, region 2 contain all the b's, and region 3 contain all the c's. If either v or y crosses numbered regions, pump in once. The resulting string will not be in L because it will violate the form constraint. We consider the remaining cases for where nonempty v and y can occur:

(1, 1): Pump out once. This reduces the number of a's and thus the *min* of the number of a's and c's. But the number of b's is unchanged so it is greater than that minimum.
(2, 2): Pump in once. The *min* of the number of a's and c's is unchanged. But the number of b's is increased and so it is greater than that minimum.
(3, 3): Same argument as (1, 1) but reduces the number of c's.
(1, 2), (2, 3): Pump in once. The *min* of the number of a's and c's is unchanged. But the number of b's is increased and so it is greater than that minimum.
(1, 3): Not possible since $|vxy|$ must be less than or equal to k.

g) $\{xyx^R : x \in \{0, 1\}^+ \text{ and } y \in \{0, 1\}^*\}$.

Regular. There is no reason to let x be more than one character. So all that is required is that the string have at least two characters and the first and last must be the same. $L = (0\ (0 \cup 1)^*\ 0) \cup (1\ (0 \cup 1)^*\ 1)$.

h) $\{xwx^R : x, w \in \{a, b\}^+ \text{ and } |x| = |w|\}$.

Not context-free. If L were context-free, then $L_1 = L \cap a^*b^*a^*ab^*a^*$ would also be context-free. But we show that it is not by pumping. Let $w =$

a^{k+1}	b^{k+1}	a^{k+1}	b^k	a	b^{k+1}	a^{k+1}
1	2	3	4	5	6	7

We break w into regions as shown above. If either v or y from the pumping theorem crosses numbered regions, pump in once. The resulting string will not be in L_1 because it will violate the form constraint. If either v or y includes region 5, pump in once and the resulting string will not be in L_1 because it will violate the form constraint. If $|vy|$ is not divisible by 3, pump in once. The resulting string will not be able to be cut into three pieces of equal length and so it is not in L_1. We now consider the other ways in which nonempty v and y could occur:
(1, 1), (2, 2), (1, 2): Pump in twice. One a and at least one b move into the last third of the string. So the first third is $a^p b^q$, for some p and q, while the last third is $b^i a^j b^k$, for some i, j, and k. So the last third is not the reverse of the first third.
(6. 6), (7, 7), (6, 7): Pump in twice. At least two a's will move into the first third of the string. So the first third is $a^p b^q a^r$, for some p, q, and r, while the last third is $a^j b^k$, for some j and k. So the last third is not the reverse of the first third.
(3, 3), (4, 4), (3, 4): Pump in twice. At least two a's will move into the first third of the string and one a and at least one b will move into the last third. So the first third is $a^p b^q a^r$, for some p, q, and r, while the last third is $b^i a^j b^k$, for some i, j, and k. So the last third is not the reverse of the first third.
(2, 3): One a and at least one b will move into the last third of the string, so it will be $b^i a^j b^k$, for some i, j, and k. Depending on the length of v relative to the length of y, the first third will be either $a^p b^q a^r$, for some p, q, and r or $a^p b^q$, for some p and q. In either case, the last third is not the reverse of the first third.
(4, 6): Pump in twice. At least two a's will move into the first third of the string. So the first third is $a^p b^q a^r$, for some p, q, and r. Depending on the length of v relative to the length of y, the last third will be $a^j b^k$, for some j and k or $b^i a^j b^k$, for some i, j, and k. In either case, the last third is not the reverse of the first third.
All remaining cases are impossible since $|vxy|$ must be less than or equal to k.

3) Let $L = \{a^n b^m c^n d^m : n, m \geq 1\}$. L is interesting because of its similarity to a useful fragment of a typical programming language in which one must declare procedures before they can be invoked. The procedure declarations include a list of the formal parameters. So now imagine that the characters in a^n correspond to the formal parameter list in the declaration of procedure 1. The characters in b^m correspond to the formal parameter list in the declaration of procedure 2. Then the characters in c^n and d^m correspond to the parameter lists in an invocation of procedure 1 and procedure 2 respectively, with the requirement that the number of parameters in the invocations match the number of parameters in the declarations. Show that L is not context-free.

Not context-free. We prove it using the Pumping Theorem. Let $w = a^k b^k c^k d^k$.
1 | 2 | 3 | 4

If either v or y from the pumping theorem crosses numbered regions, pump in once. The resulting string will not be in L because the letters will be out of order. We now consider the other ways in which nonempty v and y could occur:
(1, 1), (3, 3) Pump in once. The a and c regions will no longer be of equal length.
(2, 2), (4, 4) Pump in once. The b and d regions will no longer be of equal length.
(1, 2), (3, 4), (2, 3) Pump in once. Neither the a and c regions nor the b and d regions will be of equal length.
(1, 3), (1, 4), (2, 4) Not possible since $|vxy| \leq k$.
So L is not context-free.

6) Let $L_1 = L_2 \cap L_3$.

a) Show values for L_1, L_2, and L_3, such that L_1 is context-free but neither L_2 nor L_3 is.

Let: $L_1 = \{a^n b^n : n \geq 0\}$.
$L_2 = \{a^n b^n c^j : j \leq n\}$.
$L_3 = \{a^n b^n c^j : j = 0 \text{ or } j > n\}$.

b) Show values for L_1, L_2, and L_3, such that L_2 is context-free but neither L_1 nor L_3 is.

Let: $L_2 = a^*$.
$L_1 = \{a^n : n \text{ is prime}\}$.
$L_3 = \{a^n : n \text{ is prime}\}$.

15) Let $L_1 = \{a^n b^m : n \geq m\}$. Let $R_1 = \{(a \cup b)^* : \text{there is an odd number of a's and an even number of b's}\}$. Use the construction described in the book to build a PDA that accepts $L_1 \cap R_1$.

We start with M_1 and M_2, then build M_3:
M_1, which accepts $L_1 = (\{1, 2\}, \{a, b\}, \{a\}, \Delta, 1, \{2\})$, $\Delta =$ ((1, a, ε), (1, a))
((1, b, a), (2, ε))
((1, ε, ε), (2, ε)
((2, b, a), (2, ε)
((2, ε, a), (2, ε)

M_2, which accepts R_1 = ({1, 2, 3, 4}, {a, b}, δ, 1, {2}), δ =

(1, a, 2)
(1, b, 3)
(2, a, 1)
(2, b, 4)
(3, a, 4)
(3, b, 1)
(4, a, 3)
(4, b, 2)

M_3, which accepts $L_1 \cap R_1$ = ({(1, 1), (1, 2), (1, 3), (1, 4), (2, 1), (2, 2,), (2, 3), (2, 4)}, {a, b}, {a}, Δ, (1,1), {(2,2)}), Δ =

(((1, 1), a, ε), ((1, 2), a))
(((1, 1), b, a), ((2, 3), ε))
(((1, 2), a, ε), ((1, 1), a))
(((1, 2), b, a), ((2, 4), ε))
(((1, 3), a, ε), ((1, 4), a))
(((1, 3), b, a), ((2, 1), ε))
(((1, 4), a, ε), ((1, 3), a))
(((1, 4), b, a), ((2, 2), ε))

(((2, 1), b, a), ((2, 3), ε))
(((2, 2), b, a), ((2, 4), ε))
(((2, 3), b, a), ((2, 1), ε))
(((2, 4), b, a), ((2, 2), ε))

(((1, 1), ε, ε), ((2, 1), ε))
(((1, 2), ε, ε), ((2, 2), ε))
(((1, 3), ε, ε), ((2, 3), ε))
(((1, 4), ε, ε), ((2, 4), ε))
(((2, 1), ε, a), ((2, 1), ε))
(((2, 2), ε, a), ((2, 2), ε))
(((2, 3), ε, a), ((2, 3), ε))
(((2, 4), ε, a), ((2, 4), ε))

4 Decision Procedures for Context-Free Languages

1) Give a decision procedure to answer each of the following questions:
   a) Given a regular expression α and a PDA M, is the language accepted by M a subset of the language generated by α?

Observe that this is true iff $L(M) \cap \neg L(\alpha) = \varnothing$. So the following procedure answers the question:

1. From α, build an FSM M^* so that $L(M^*) = L(\alpha)$.
2. From M^*, build a new FSM M^{**} so that $L(M^{**}) = \neg L(M^*)$.
3. From M and M^{**}, build a PDA M^{***} that accepts $L(M) \cap L(M^{**})$
4. If $L(M^{***})$ is empty, return *True*, else return *False*.

   b) Given a context-free grammar G and two strings s_1 and s_2, does G generate s_1s_2?

1. Convert G to Chomsky Normal Form.
2. Try all derivations in G of length up to $2|s_1s_2|$. If any of them generates s_1s_2, return *True*, else return *False*.

   c) Given a context-free grammar G, does G generate at least three strings?

To construct a decision procedure for this problem, we exploit the same idea that we used as the basis of the Pumping Theorem. Let b be the branching factor of G and let n be the number of G's nonterminals. Then the longest string that G can generate and assign a parse tree with no duplicated nonterminals on any path has length no more than b^n. If G can generate even one string of length greater than that, then that string can be pumped. So G generates an infinite number of strings and thus at least three. Further, if there is at least one such long string, there must also be a shorter one (the result of pumping out from it). We also know that $|vxy| \leq b^{n+1}$. So if G generates any string of length greater than $b^n + b^{n+1}$, then it must also generate at least one whose length is between b^n and b^{n+1} (because we can keep pumping out until we get such a shorter string). So the following algorithm decides the question:

1. Examine G and determine b and n.
2. Set *count* to 0.
3. For every string w in Σ_G^* such that $|w| \leq b^n$ do:
   3.1. If *decideCFL*(G, w) then *count* = *count* + 1.
4. If *count* = 0 then return *False*.
5. If *count* $\geq$ 3 then return *True*.
6. For every string w in Σ_G^* such that $b^n < |w| \leq b^n + b^{n+1}$ do:
   6.1. If *decideCFL*(G, w) then return *True*.
7. Return *False*.

   d) Given a context-free grammar G, does G generate any even length strings?

1. Use *CFGtoPDAtopdown*(G) to build a PDA P that accepts $L(G)$.
2. Build an FSM E that accepts all even length strings over the alphabet Σ_G.
3. Use *intersectPDAandFSM*(P, E) to build a PDA P^* that accepts $L(G) \cap L(E)$.
4. If *decideCFLempty*(P^*) returns *True* then return *False*. Otherwise, return *True*.

   e) Given a regular grammar G, is $L(G)$ context-free?

1. Return *True* (since every regular language is context-free).