CS 341 Homework 11
Context-Free Grammars

1. Consider the grammar G = (V, Σ, R, S), where

$V = \{a, b, S, A\}$,
$\Sigma = \{a, b\}$,
$R = \{ \quad S \to AA,$
$\quad A \to AAA,$
$\quad A \to a,$
$\quad A \to bA,$
$\quad A \to Ab \ \}.$

(a) Which strings of L(G) can be produced by derivations of four or fewer steps?
(b) Give at least four distinct derivations for the string babbab.
(c) For any m, n, p > 0, describe a derivation in G of the string $b^m a b^n a b^p$.

2. Construct context-free grammars that generate each of these languages:
(a) $\{wcw^R : w \in \{a, b\}^*\}$
(b) $\{ww^R : w \in \{a, b\}^*\}$
(c) $\{w \in \{a, b\}^* : w = w^R\}$

3. Consider the alphabet $\Sigma = \{a, b, (,), \cup, *, \varnothing\}$. Construct a context-free grammar that generates all strings in Σ^* that are regular expressions over {a, b}.

4. Let G be a context-free grammar and let k > 0. We let $L_k(G) \subseteq L(G)$ be the set of all strings that have a derivation in G with k or fewer steps.
(a) What is $L_5(G)$, where $G = (\{S, (,)\}, \{(,)\}, \{S \to \varepsilon, S \to SS, S \to (S)\})$?
(b) Show that, for all context-free grammars G and all k > 0, $L_k(G)$ is finite.

5. Let G = (V, Σ, R, S), where

$V = \{a, b, S\}$,
$\Sigma = \{a, b\}$,
$R = \{ \quad S \to aSb,$
$\quad S \to aSa,$
$\quad S \to bSa,$
$\quad S \to bSb,$
$\quad S \to \varepsilon \ \}.$

Show that L(G) is regular.

6. A program in a procedural programming language, such as C or Java, consists of a list of statements, where each statement is one of several types, such as:
(1) assignment statement, of the form id := E, where E is any arithmetic expression (generated by the grammar using T and F that we presented in class).
(2) conditional statement, e.g., "if E < E then statement", or while statement , e.g. "while E < E do statement".
(3) goto statement; furthermore each statement could be preceded by a label.
(4) compound statement, i.e., many statements preceded by a begin, followed by an end, and separated by ";".

Give a context-free grammar that generates all possible statements in the simplified programming language described above.

7. Show that the following languages are context free by exhibiting context-free grammars generating each:
(a) $\{a^m b^n : m \geq n\}$

(b) $\{a^m b^n c^p d^q : m + n = p + q\}$
(c) $\{w \in \{a, b\}^* : w$ has twice as many b's as a's$\}$
(d) $\{uawb : u, w \in \{a, b\}^*, |u| = |w|\}$

8. Let $\Sigma = \{a, b, c\}$. Let L be the language of prefix arithmetic defined as follows:

(i) any member of Σ is a well-formed expression (wff).
(ii) if α and β are any wff's, then so are $A\alpha\beta$, $S\alpha\beta$, $M\alpha\beta$, and $D\alpha\beta$.
(iii) nothing else is a wff.

(One might think of A, S, M, and D as corresponding to the operators +, -, ×, /, respectively. Thus in L we could write Aab instead of the usual (a + b), and MSabDbc, instead of ((a - b) × (b/c)). Note that parentheses are unnecessary to resolve ambiguities in L.)

(a) Write a context-free grammar that generates exactly the wff's of L.
(b) Show that L is not regular.

9. Consider the language $L = \{a^m b^{2n} c^{3n} d^p : p > m$, and $m, n \geq 1\}$.
(a) What is the shortest string in L?
(b) Write a context-free grammar to generate L.

Solutions

1. (a) We can do an exhaustive search of all derivations of length no more than 4:

$S \Rightarrow AA \Rightarrow aA \Rightarrow aa$
$S \Rightarrow AA \Rightarrow aA \Rightarrow abA \Rightarrow aba$
$S \Rightarrow AA \Rightarrow aA \Rightarrow aAb \Rightarrow aab$
$S \Rightarrow AA \Rightarrow bAA \Rightarrow baA \Rightarrow baa$
$S \Rightarrow AA \Rightarrow bAA \Rightarrow bAa \Rightarrow baa$
$S \Rightarrow AA \Rightarrow AbA \Rightarrow abA \Rightarrow aba$
$S \Rightarrow AA \Rightarrow AbA \Rightarrow Aba \Rightarrow aba$
$S \Rightarrow AA \Rightarrow Aa \Rightarrow aa$
$S \Rightarrow AA \Rightarrow Aa \Rightarrow bAa \Rightarrow baa$
$S \Rightarrow AA \Rightarrow Aa \Rightarrow Aba \Rightarrow aba$
$S \Rightarrow AA \Rightarrow AbA \Rightarrow abA \Rightarrow aba$
$S \Rightarrow AA \Rightarrow AbA \Rightarrow Aba \Rightarrow aba$
$S \Rightarrow AA \Rightarrow AAb \Rightarrow aAb \Rightarrow aab$
$S \Rightarrow AA \Rightarrow AAb \Rightarrow Aab \Rightarrow aab$

Many of these correspond to the same parse trees, just applying the rules in different orders. In any case, the strings that can be generated are: aa, aab, aba, baa.

(b) Notice that $A \Rightarrow bA \Rightarrow bAb \Rightarrow bab$, and also that $A \Rightarrow Ab \Rightarrow bAb \Rightarrow bab$. This suggests 8 distinct derivations:

$S \Rightarrow AA \Rightarrow AbA \Rightarrow AbAb \Rightarrow Abab \Rightarrow^*$ babbab
$S \Rightarrow AA \Rightarrow AAb \Rightarrow AbAb \Rightarrow Abab \Rightarrow^*$ babbab
$S \Rightarrow AA \Rightarrow bAA \Rightarrow bAbA \Rightarrow babA \Rightarrow^*$ babbab
$S \Rightarrow AA \Rightarrow AbA \Rightarrow bAbA \Rightarrow babA \Rightarrow^*$ babbab

Where each of these four has 2 ways to reach babbab in the last steps. And, of course, one could interleave the productions rather than doing all of the first A, then all of the second A, or vice versa.

(c) This is a matter of formally describing a sequence of applications of the rules in terms of m, n, p that will produce the string $b^m a b^n a b^p$.

S
$\Rightarrow$ /* by rule $S \rightarrow AA$ */

$$\Rightarrow^* \text{ /* by m applications of rule } A \rightarrow bA \text{ */} \quad \begin{matrix} AA \\ b^mAA \end{matrix}$$

$$\Rightarrow \text{ /* by rule } A \rightarrow a \text{ */} \quad b^maA$$

$$\Rightarrow^* \text{ /* by n applications of rule } A \rightarrow bA \text{ */} \quad b^mab^nA$$

$$\Rightarrow^* \text{ by p applications of rule } A \rightarrow Ab \text{ */} \quad b^mab^nAb^p$$

$$\Rightarrow \text{ /* by rule } A \rightarrow a \text{ */} \quad b^mab^nab^p$$

Clearly this derivation (and some variations on it) produce $b^mab^nab^p$ for each m, n, p.

2. (a) G = (V, Σ, R, S) with V = {S, a, b, c}, Σ = {a, b, c}, R = {

$S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow c$ }.

(b) Same as (a) except remove c from V and Σ and replace the last rule, $S \rightarrow c$, by $S \rightarrow \varepsilon$.

(c) This language very similar to the language of (b). (b) was all even length palindromes; this is all palindromes. We can use the same grammar as (b) except that we must add two rules:

$S \rightarrow a$

$S \rightarrow b$

3. This is easy. Recall the inductive definition of regular expressions that was given in class :

1. ∅ and each member of Σ is a regular expression.
2. If α , β are regular expressions, then so is αβ
3. If α , β are regular expressions, then so is α∪β.
4. If α is a regular expression, then so is α*.
5. If α is a regular expression, then so is (α).
6. Nothing else is a regular expression.

This definition provides the basis for a grammar for regular expressions:

G = (V, Σ, R, S) with V = {S, a, b, (,), ∪, *, ∅}, Σ = { a, b, (,), ∪, *, ∅}, R= {

$S \rightarrow \varnothing$ /* part of rule 1, above

$S \rightarrow a$ /* "

$S \rightarrow b$ /* "

$S \rightarrow SS$ /* rule 2

$S \rightarrow S \cup S$ /* rule 3

$S \rightarrow S^*$ /* rule 4

$S \rightarrow (S)$ /* rule 5 }

4. (a) We omit derivations that don't produce strings in L (i.e, they still contain nonterminals).

$L_1 : S \Rightarrow \varepsilon$

$L_2 : S \Rightarrow (S) \Rightarrow ()$

$L_3 : S \Rightarrow SS \Rightarrow \varepsilon S \Rightarrow \varepsilon$

$S \Rightarrow (S) \Rightarrow ((S)) \Rightarrow (())$

$L_4 : S \Rightarrow SS \Rightarrow (S)S \Rightarrow ()S \Rightarrow ()$

$S \Rightarrow SS \Rightarrow S(S) \Rightarrow (S) \Rightarrow ()$

$S \Rightarrow (S) \Rightarrow ((S)) \Rightarrow (((S))) \Rightarrow ((()))$

$L_5 : S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow ()(S) \Rightarrow ()()$

$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S \Rightarrow (())$

$S \Rightarrow (S) \Rightarrow ((S)) \Rightarrow (((S))) \Rightarrow ((((S)))) \Rightarrow ((()))$

So $L_5 = \{\varepsilon, (), (()), ((())), (((()))), ()()\ \}$

(b) We can give a (weak) upper bound on the number of strings in $L_K(G)$. Let P be the number of rules in G and let N be the largest number of nonterminals on the right hand side of any rule in G. For the first derivation step, we start with S and have P choices of derivations to take. So at most P strings can be generated. (Generally there will be many fewer, since many rules may not apply, but we're only producing an upper bound here, so that's okay.) At the second step, we may have P strings to begin with (any one of the ones produced in the first step), each of them may have up to N nonterminals that we could choose to expand, and each nonterminal could potentially be expanded in P ways. So the number of strings that can be produced is P×N×P. Note that many of them aren't strings in L since they may still contain nonterminals, but this number is an upper bound on the number of strings in L that can be produced. At the third derivation step, each of those strings may again have N nonterminals that can be expanded and P ways to expand each. In general, an upper bound on the number of strings produced after K derivation steps is $P^K N^{(K-1)}$, which is clearly finite. The key here is that there is a finite number of rules and that each rule produces a string of finite length.

5. We will show that L(G) is precisely the set of all even length strings in {a, b}*. But we know that that language is regular. QED.

First we show that only even length strings are generated by G. This is trivial. Every rule that generates any terminal characters generates two. So only strings of even length can be generated.

Now we must show that all strings of even length are produced. This we do by induction on the length of the strings:
Base case: $\varepsilon \in L_G$ (by application of the last rule). So we generate the only string of length 0.

Induction hypothesis: All even length strings of length ≤ N (for even N) can be generated from S.

Induction step: We need to show that any string of length N+2 can be generated. Any string w of length N + 2 ($N \geq 0$) can be rewritten as xyz, where x and z are single characters and |y| = N. By the induction hypothesis, we know that all values of y can be generated from S. We now consider all possible combinations of values for x and z that can be used to create w. There are four, and the first four rules in the grammar generate, for any string T derivable from S, the four strings that contain T plus a single character appended at the beginning and at the end. Thus all possible strings w of length N+2 can be generated.

6. Since we already have a grammar for expressions (E), we'll just use E in this grammar and treat it as though it were a terminal symbol. Of course, what we really have to do is to combine this grammar with the one for E. As we did in our grammar for E, we'll use the terminal string id to stand for any identifier.
G = (V, Σ, R, S), where V = {S, U, C, L, T, E, :, = <, >, ;, a-z, id}, Σ = { :, = <, >, ;, a-z, id}, and
R= {

S → L U	/* a statement can be a label followed by an unlabeled statement
S → U	/* or a statement can be just an unlabeled statement. We need to make the distinction between S and U if we want to prevent a statement from being preceded by an arbitrary number of labels.
U → id := E	/* assignment statement
U → if E T E then S	/* if statement
U → while E T E do S	/* while statement
U → goto L	/* goto statement
U → begin S; S end	/* compound statement
L → id	/* a label is just an identifier

T→ < | > | = /* we use T to stand for a test operator. We introduce the | (or) notation here for convenience. }

There's one problem we haven't addressed here. We have not guaranteed that every label that appears after a goto statement actually appears in the program. In general, this cannot be done with a context-free grammar.

7. (a) $L = \{a^m b^m : m \geq n\}$. This one is very similar to Example 8 in Supplementary Materials: Context-Free Languages and Pushdown Automata: Designing Context-Free Grammars. The only difference is that in that case, $m \leq n$. So you can use any of the ideas presented there to solve this problem.

(b) $L = \{a^m b^n c^p d^q : m + n = p + q\}$. This one is somewhat like **(a)**: 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's 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:

G = ({S, T, U, V, a, b, c, d}, {a, b, c, d}, R, S), where

R = {S → aSd, S → T, S → U, T → aTc, T → V, U → bUd, U → V, V → bVc, V → ε}

Every derivation will use symbols S, T, V in sequence or S, U, V in sequence. As a quick check for fencepost errors, note that the shortest string in L is ε, which is indeed generated by the grammar. (And we do not need any rules S → ε or T → ε.)

How do we know this grammar works? Notice that any string for which m = q has two distinct derivations:

$S \Rightarrow^* a^m S d^m \Rightarrow a^m T d^m \Rightarrow a^m V d^m \Rightarrow a^m b^n c^p d^q$, and

$S \Rightarrow^* a^m S d^m \Rightarrow a^m U d^m \Rightarrow a^m V d^m \Rightarrow a^m b^n c^p d^q$

Every string $a^m b^n c^p d^q \in L$ for which $m \geq q$ has a derivation:

S

⇒ /* by q application of rule S → aSd */

$a^q S d^q$

⇒ /* by rule S → T */

$a^q T d^q$

⇒ /* by m - q application of rule rule T → aTc */

$a^q a^{m-q} T c^{m-q} d^q = a^m T c^{m-q} d^q$

⇒ /* by rule T → V */

$a^m V c^{m-q} d^q$

⇒ /* by n = p - (m - q) applications of rule V → bVc */

$a^m b^n V c^{p-(m-q)} c^{m-q} d^q = a^m b^n V c^p d^q$

⇒ /* by rule V → ε

$a^m b^n c^p d^q$

For the other case ($m \leq q$), you can show the corresponding derivation. So every string in L is generated by G. And it can be shown that no string not in L is generated by G.

(c) $L = \{w \in \{a, b\}^* : w$ has twice as many b's as a's$\}$. This one is sort of tricky. Why? Because L doesn't care about the order in which the a's and b's occur. But grammars do. One solution is:

G = ({S, a, b}, {a, b}, R, S), where R = {S → SaSbSbS, S → SbSaSbS, S → SbSbSaS, S → ε}

Try some examples to convince yourself that this grammar works. Why does it work? Notice that all the rules for S preserve the invariant that there are twice as many b's as a's. So we're guaranteed not to generate any strings that aren't in L. Now we just have to worry about generating all the strings that are in L. The first three rules handle the three possible orders in which the symbols b,b, and a can occur.

Another approach you could take is to build a pushdown automaton for L and then derive a grammar from it. This may be easier simply because PDA's are good at counting. But deriving the grammar isn't trivial either. If you had a hard time with this one, don't worry.

(d) L = {uawb : u, w $\in$ {a, b}*, |u| = |w|}. This one fools some people since you might think that the a and b are correlated somehow. But consider the simpler language L' = {uaw : u, w $\in$ {a, b}*, |u| = |w|}. This one seems easier. We just need to generate u and w in parallel, keeping something in the middle that will turn into a. Now back to L: L is just L' with b tacked on the end. So a grammar for L is:

G = ({S, T, a, b}, {a, b}, R, S), where R = {S $\rightarrow$ Tb, T $\rightarrow$ aTa, T $\rightarrow$ aTb, T $\rightarrow$ bTa, T $\rightarrow$ bTb, T $\rightarrow$ a}.

8. (a) G = ({S, A, M, D, F, a, b, c}. {A, M, D, S, a, b, c}, R, S), where R = {

F $\rightarrow$ a	F $\rightarrow$ AFF
F $\rightarrow$ b	F $\rightarrow$ SFF
F $\rightarrow$ c	F $\rightarrow$ MFF
	F $\rightarrow$ DFF }

(b) First, we let L' = L $\cap$ A*a*. L' = $\{A^n a^{n+1} : n \geq 0\}$. L' can easily be shown to be nonregular using the Pumping Theorem, so, since the regular languages are closed under intersection, L must not be regular.

9. (a) abbcccdd

(b) G = ({S, X, Y, a, b, c, d}, {a, b, c, d}, R, S), where R is either:

(S $\rightarrow$ aXdd, X $\rightarrow$ Xd, X $\rightarrow$ aXd, X $\rightarrow$ bbYccc, Y $\rightarrow$ bbYccc, Y $\rightarrow$ ε), or

(S $\rightarrow$ aSd, S $\rightarrow$ Sd, S $\rightarrow$ aMdd, M $\rightarrow$ bbccc, M $\rightarrow$ bbMccc)