Problem 1 (5.1.1 (b), (c), and (d) from the textbook). Design context-free grammars for the following languages.

(1) The set \{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}, that is, the set of strings of a’s followed by b’s followed by c’s, such that there are either a different number of a’s and b’s or different number of b’s and c’s, or both.

(2) The set of all strings of a’s and b’s that are not of the form ww, that is, not equal to any string repeated.

(3) The set of all strings with twice as many 0’s as 1’s.

Problem 2 (5.1.4 from the textbook). A CFG is said to be right-linear if each production body has at most one variable, and that variable is at the right end. That is, all productions of a right-linear grammar are of the form \(A \rightarrow wB\) or \(A \rightarrow w\), where \(A\) and \(B\) are variables and \(w\) some string of zero or more terminals.

(a) Show that every right-linear grammar generates a regular language. Hint: Construct an \(\epsilon\)-NFA that simulates leftmost derivations, using its state to represent the lone variable in the current left-sentential form.

(b) Show that every regular language has a right-linear grammar. Hint: Start with a DFA and let the variables of the grammar represent states.

Problem 3 (5.1.5 from the textbook). Let \(T = \{0, 1, (, ), +, *, 0, e\}\). We may think of \(T\) as the set of symbols used by regular expressions over alphabet \(\{0, 1\}\); the only difference is that we use \(e\) for symbol \(\epsilon\), to avoid potential confusion in what follows. Your task is to design a CFG with set of terminals \(T\) that generates exactly the regular expressions with alphabet \(\{0, 1\}\).

Problem 4 (5.2.3 from the textbook). Let \(G\) be a CFG (possibly with productions which have \(\epsilon\) as the right side). If \(w \neq \epsilon\) is in \(L(G)\), the length of \(w\) is \(n\), and \(w\) has a derivation of \(m\) steps, then show that the parse tree for \(w\) may have as many as \(n + 2m - 1\) nodes, but no more.

Problem 5 (5.3.2 from the textbook). Consider the set of all strings of balanced parentheses of two types, round and square. An example of where these strings come from is as follows. If we take expressions in programming language C, which use round parentheses for grouping and for arguments of functions calls, and use square brackets for array indexes, and drop out everything but the parentheses, we get all strings of balanced parentheses of these two types. For example,

\[f(a[i] *(b[i][j], c[g(x)]), d[i])\]

becomes the balanced-parenthesis string \(\{(\[] [][](())[]\}\). Design a grammar for all and only the strings of round and square parentheses that are balanced.