Quiz EL502

Context-Free Grammars and Pushdown Automata

Lucas Monteiro Nogueira

Note: In the following problems, S is the start symbol of any given context-free grammar, unless stated otherwise.

PROBLEMS

Problem 1

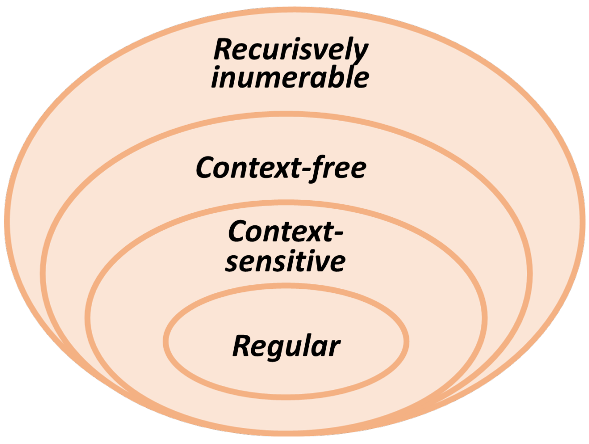
Regarding the theory of grammars and pushdown automata, true or false?

1.( ) Context-free languages are closed under union, concatenation, intersection, and Kleene closure.

2.( ) One important operation in languages is the cycle operation *cyc*. Roughly speaking, *cyc* sends every string to the set of all its cyclic shifts, or conjugates. Importantly, the class of regular languages is closed under the *cyc* operation. Furthermore, it can be shown that if L is regular, then so is $cyc(L)$.

3.( ) Finite state automata are to regular languages as nondeterministic pushdown automata are to context-free languages.

4.( ) The following diagram correctly illustrates the set inclusions described by the Chomsky hierarchy of languages.



5.( ) The context-free grammar G with the following rules is unambiguous.

$$S \to aA$$
$$A \to BA \mid a$$
$$B \to bS \mid cS$$

6.( ) If G is a context-free grammar in Chomsky normal form, then for any string $w \in L(G)$ of length $n \geq 1$, exactly $2n + 1$ steps are required for any derivation of w.

7.( ) The grammar below is a simple or s-grammar.

$$S \to aS \mid bSS \mid aSS \mid c$$

8.( ) Every grammar in Greibach normal form is also a simple or s-grammar.

Problem 2

Construct the string `0100110` from the following grammar by using

Problem 2.1: Leftmost derivation

Problem 2.2: Rightmost derivation

$$S \to 0S \mid 1PP$$
$$P \to 0 \mid 1P \mid 0Q$$
$$Q \to 1 \mid 0QQ$$

Problem 2.3: Draw a parse tree for the leftmost derivation of the string in question.

Problem 3

Construct the string *abbbb* from the following grammar by using

Problem 3.1: Leftmost derivation

Problem 3.2: Rightmost derivation

$$S \to aAB$$
$$A \to bBb$$
$$B \to A \mid \varepsilon$$

Problem 4

Using the following grammar, draw the parse tree that corresponds to the leftmost derivation of string 11001010.

$$S \to 1N \mid 0M$$
$$M \to 1 \mid 1S \mid 0MM$$
$$N \to 0 \mid 0S \mid 1NN$$

Problem 5

Consider the grammar

$$S \to aS \mid aSbS \mid \varepsilon$$

Problem 5.1: This grammar is ambiguous. Show in particular that the string *aab* has two parse trees for leftmost derivation.

Problem 5.2: Find an unambiguous grammar for the language generated by the grammar above.

Problem 6 (Hopcroft *et al.*, 2001)

Let $T = \{0, 1, (,), +, \star, \emptyset, \varepsilon\}$. We may think of T as the set of symbols used by regular expressions over alphabet $\{0,1\}$. Your task is to design a context-free grammar with set of terminals T that generates exactly the regular expressions with alphabet $\{0,1\}$. Is the grammar you designed in the previous part unambiguous? If not, redesign it to be unambiguous.

Problem 7

Problem 7.1: Consider the context-free grammar $G_1 = (\{S\}, \{0,1\}, \{S \to \varepsilon, S \to 0S1\}, S)$. What is the language generated by this grammar?

Problem 7.2: Consider the context-free grammar $G_2 = (\{S, A, B\}, \{a,b\}, R, S)$, where R consists of the following rules,

$$S \to ABA$$
$$A \to a \mid bb$$
$$B \to bS \mid \varepsilon$$

What is the language generated by this grammar?

Problem 8

Give context-free grammars that generate the following languages. In all parts, the alphabet is $\Sigma = \{0,1\}$.

Problem 8.1: The empty set

Problem 8.2: $\{w \mid w$ starts and ends with the same symbol$\}$

Problem 8.3: $\{w \mid$ the length of w is odd$\}$

Problem 8.4: $\{w \mid w = w^R$, that is, w is a palindrome$\}$

Problem 9

Give informal descriptions of pushdown automata used to implement the four languages developed in Problem 8.

Problem 10

Provide grammars that can be used to generate the following languages.

Problem 10.1: The complement of the language $\{a^n b^n \mid n \geq 0\}$

Problem 10.2: $\{x_1\#x_2\# \ldots \#x_k \mid k \geq 1$, each $x_i \in \{a,b\}^*$, and for some i and j, $x_i = x_j^R\}$

Problem 11

Problem 11.1: Give a context-free grammar that generates the language

$$A = \{a^i b^j c^k \mid i = j \text{ or } j = k \text{ where } i, j, k \geq 0\}$$

Is your grammar ambiguous? Why or why not?

Problem 11.2: Give an informal description of a pushdown automaton that recognizes the language A specified in the previous part.

Problem 12

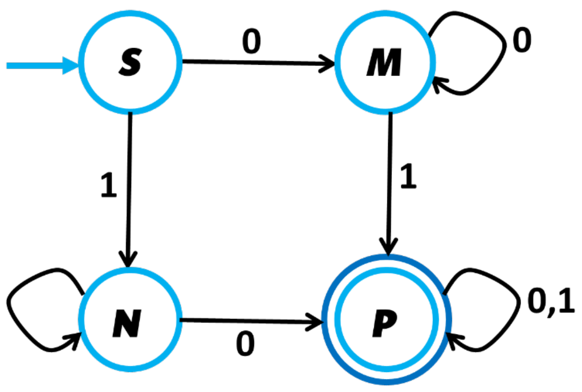
Consider the regular grammar

$$G = (\{S,A\}, \{0,1\}, \{S \to 0A, A \to 1A \mid 0 \mid 1S\}, S)$$

Construct a pushdown automaton for the language generated by grammar G.

Problem 13

Construct a context-free grammar that generates the regular language accepted by the discrete finite automaton illustrated below.



Problem 14 (Kumar, 2010)

Problem 14.1: Consider the pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, s, F) = (\{q, p\}, \{a, b, c\}, \{a, b\}, \delta, q, \{p\})$ (note that we are not using the 7-tuple notation, as we have omitted the initial stack symbol Z_0), whose transition function is described by the following rules. Describe the language $L(M)$.

$\delta(q, a, \varepsilon) = \{(q,a)\}$	$\delta(p, a, a) = \{(p, \varepsilon)\}$
$\delta(q, b, \varepsilon) = \{(q,b)\}$	$\delta(p, b, b) = \{(p, \varepsilon)\}$
$\delta(q, c, \varepsilon) = \{(p,\varepsilon)\}$	

Problem 14.2: Propose transition rules for a pushdown automaton that accepts all strings over $\{a,b\}$ that contain an equal number of a's and b's; the corresponding language is of course

$$L = \{s \in (a,b)^* \mid n_a(s) = n_b(s)\} \text{ by null store.}$$

Problem 14.3: Propose transition rules for a pushdown automaton that accepts all strings over $\{a,b\}$ that describes the language

$$L = \{x \in (a,b)^* \mid \text{number of } a\text{'s is greater than number of } b\text{'s}\}$$

Problem 14.4: Design a pushdown automaton for the language

$$L = \{s \in (a,b)^* \mid n_a(s) \neq n_b(s)\}$$

That is, language L describes strings s in $\{a, b\}$ such that the number of a's is not equal to the number of b's.

Problem 14.5: Design a pushdown automaton for language $L = \{a^n b^{3n} \mid n \geq 0\}$.

Problem 14.6: Design a pushdown automaton for language $L = \{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}$.

Problem 15 (Kumar, 2010)

Reduce context-free grammar G to Chomsky normal form.

$$S \to aAC$$
$$A \to aB \mid bAB$$
$$B \to b$$
$$C \to c$$

SOLUTIONS

P.1 Solution

1. False. Context-free languages are closed under union, concatenation, and Kleene closure, but not under intersection or complement. Hopcroft's textbook provides a quick example of nonclosure under intersection. The language L below is not context-free.

$$L = \{0^n 1^n 2^n \mid n \geq 1\}$$

In contrast, the two following languages are in fact context-free,

$$L_1 = \{0^n 1^n 2^i \mid n \geq 1, i \geq 1\}$$

$$L_2 = \{0^i 1^n 2^n \mid n \geq 1, i \geq 1\}$$

A grammar for L_1 is described below.

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow 0A1 \mid 01 \\ B &\rightarrow 2B \mid 2 \end{aligned}$$

In this grammar, A generates all strings of the form 0^n1^n, and B generates all strings of 2's. A grammar for L_2 is

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow 0A \mid 0 \\ B &\rightarrow 1B2 \mid 12 \end{aligned}$$

This grammar works similarly, but with A generating any string of 0's, and B generating matching strings of 1's and 2's.

However, $L = L_1 \cap L_2$. To see why, observe that L_1 requires that there be the same number of 0's and 1's, while L_2 requires the numbers of 1's and 2's to be equal. A string in both languages must have equal numbers of all three symbols and thus be in L. If the CFLs were closed under intersection, then we could prove the false statement that L is context-free. We conclude by contradiction that the CFLs are not closed under intersection.

2. True. Letting L be a regular language and $cyc(L)$ the cycled language such that

$$cyc(L) \coloneqq \{x_1x_2 : x_2x_1 \in L\}$$

it follows that $cyc(L)$ is also regular.

3. True. Nothing to add here.

4. False. For the illustration to be correct, "context-sensitive" and "context-free" should switch places.

5. True. Since there is only one S-rule, we may write the lookahead set $LA_1(S) = LA_1(S \rightarrow aA)$. Next, it is easy to see that the two lookahead sets for B are $LA_1(B \rightarrow bS) = \{b\}$ and $LA_1(B \rightarrow cS) = \{c\}$, which are disjoint. Finally, the two lookahead sets of A are $LA_1(A \rightarrow BA) = \{b,c\}$ and $LA_1(A \rightarrow a) = \{a\}$, which are likewise disjoint. Accordingly, G is a strong $LL(1)$ grammar and can be said to be unambiguous.

6. False. For every $n \geq 1$, a derivation of string w requires $2n - 1$ steps.

7. False. A context-free grammar $G = (V, T, S, P)$ is said to be a simple grammar if all its productions are of the form $A \rightarrow ax$, where $A \in V$, $a \in T$, $x \in V^*$, and any pair (A,a) occurs at most once in P. The grammar at hand is not an s-grammar because the pair (S, a) occurs in two productions, namely $S \rightarrow aS$ and $S \rightarrow aSS$.

8. False. A context-free grammar $G = (V, T, S, P)$ is said to be in Greibach normal form if all productions have the form $A \rightarrow ax$, where $A \in V$, $a \in T$, $x \in V^*$. Unlike in the definition of s-grammar, however, there is no restriction on the number of occurrences of pairs (A, a). It follows that every s-grammar also obeys the Greibach normal form, but not every grammar in Greibach normal form is a s-grammar.

P.2 ➡ *Solution*

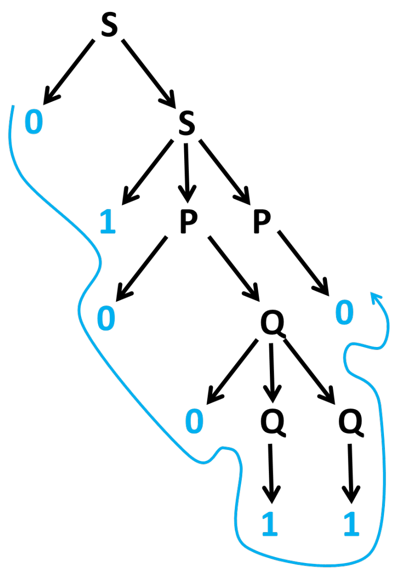
Problem 2.1: The leftmost derivation is developed below. Symbols substituted over the course of the derivation are highlighted with an underscore.

$$S \rightarrow 0S \rightarrow 01\underline{P}P \rightarrow 010\underline{Q}P \rightarrow 0100\underline{Q}QP \rightarrow 01001\underline{Q}P \rightarrow 010011\underline{P} \rightarrow \boldsymbol{0100110}$$

Problem 2.2: The rightmost derivation is shown in continuation.

$$S \rightarrow 0\underline{S} \rightarrow 01P\underline{P} \rightarrow 01\underline{P}0 \rightarrow 010Q0 \rightarrow 0100Q\underline{Q}0 \rightarrow 0100\underline{Q}10 \rightarrow \boldsymbol{0100110}$$

Problem 2.3: The parse tree for the leftmost derivation is shown on the next page.



P.3 ➡ *Solution*

Problem 3.1: The leftmost derivation is shown below. Substituted symbols are highlighted with an underscore.

$$S \to a\underline{A}B \to ab\underline{B}bB \to ab\underline{A}bB \to abb\underline{B}bbB \to abbbb\underline{B} \to \boldsymbol{abbbb}$$

Problem 3.2: The rightmost derivation is shown in continuation.

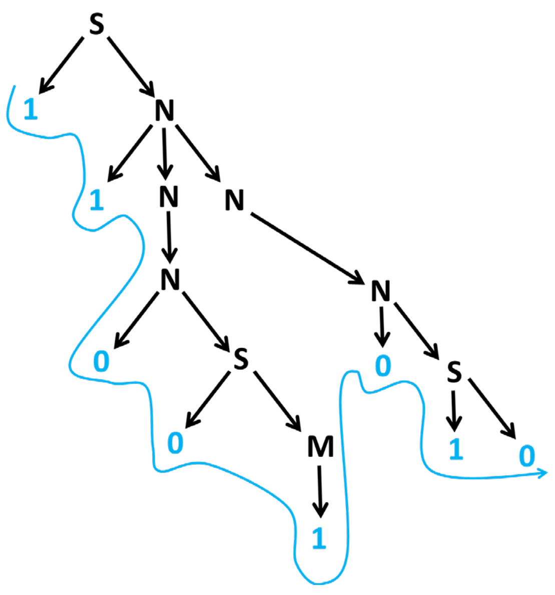
$$S \to aA\underline{B} \to a\underline{A} \to ab\underline{B}b \to ab\underline{A}b \to abb\underline{B}bb \to \boldsymbol{abbbb}$$

P.4 ➡ *Solution*

The derivation we need is shown below.

$$S \to 1\underline{N} \to 11\underline{N}N \to 110\underline{S}N \to 1100\underline{M}N \to 11001\underline{N} \to 110010\underline{S} \to 1100101\underline{N} \to 11001010$$

The parse tree is sketched next.



P.5 ➡ *Solution*

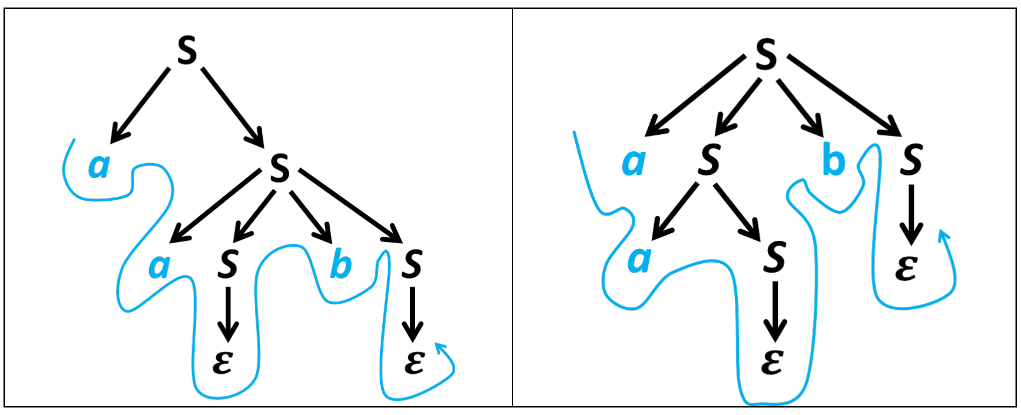
Problem 5.1: Using a leftmost derivation, we write

$$S \to aS \to aaSbS \to aabS \to aab$$

However, a leftmost derivation can also be obtained by writing

$$S \to aSbS \to aaSbS \to aabS \to aab$$

The parse trees are shown below.



Problem 5.2: The idea is to introduce another nonterminal T that cannot generate an unbalanced a. That strategy corresponds to the usual rule in programming languages that an "else" is associated with the closest previous, unmatched "then." Here, we force a b to match the previous unmatched a. Proceeding accordingly, we obtain the grammar

$$S \to aS \mid aTbS \mid \varepsilon$$
$$T \to aTbT \mid \varepsilon$$

P.6 ➡ Solution

The grammar we desire can be straightforwardly represented as

$$S \to S{+}S \mid SS \mid S^* \mid (S) \mid 0 \mid 1 \mid \emptyset \mid \varepsilon$$

The idea is that these productions for S allow any expression to be, respectively, the sum (union) of two expressions, the concatenation of two expressions, the star of an expression, a parenthesized expression, or one of the four basis cases of expressions: 0, 1, ∅, and ε. The grammar is not unambiguous. We need to have three nonterminals, corresponding to the three possible "strengths" of expressions:
1. A *factor* cannot be broken by any operator. These are the basis expressions, parenthesized expressions, and the expressions followed by one or more ⋆'s.
2. A *term* can be broken only by a ⋆. For example, consider 01, where the 0 and 1 are concatenated, but if we follow it by a ⋆, it becomes 0(1*), and the concatenation has been "broken" by the ⋆.
3. An *expression* can be broken by concatenation or ⋆, but not by +. An example is the expression 0+1. Note that if we concatenate, say, 1 or follow by a ⋆, we parse the expression 0+(11) or 0+(1⋆), and in either case the union has been broken.

Denoting *factor* by F, *term* by T, and *expression* by E, we propose the unambiguous grammar

$$F \to F^* \mid (E) \mid 0 \mid 1 \mid \emptyset \mid \varepsilon$$
$$T \to TF \mid F$$
$$E \to E{+}T \mid T$$

P.7 ➡ Solution

Problem 7.1: There is only one nonterminal symbol in G_1 and the only rule that retains that symbol is $S \to 0S1$. Thus, the derivations of G_1 have general form

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow \ldots \Rightarrow 0^n S 1^n \Rightarrow 0^n 1^n$$

Thus, the language we aim for is described as

$$\underline{L(G_1) = \{0^n 1^n \mid n \geq 0\}}$$

Problem 7.2: Since the nonterminal symbol A can only produce terminal symbols, we may delay its substitution to the end. It follows that derivations of G_2 have general form such that

$$S \Rightarrow ABA \Rightarrow AbSA \Rightarrow AbABAA \Rightarrow AbAbSAA$$
$$\Rightarrow A(bA)^2 BA^3 \Rightarrow \ldots \Rightarrow A(bA)^n BA^{n+1} \Rightarrow A(bA)^n A^{n+1}$$

Now, if we replace each A by either a or bb, we get a sentence of the form

$$(a+bb)(ba+bbb)^n (a+bb)^{n+1}$$

Thus, language $L(G_2)$ consists of all these strings with $n \geq 0$.

P.8 ➡ Solution

Problem 8.1: Doesn't get any easier than this, does it?

$$S \to S$$

Problem 8.2: The grammar we desire can be represented by the productions

$$S \to 0R0 \mid 1R \mid \varepsilon$$
$$R \to 0R \mid 1R \mid \varepsilon$$

Problem 8.3: The grammar we aim for can be represented by the productions

$$S \to 0 \mid 1 \mid 00S \mid 01S \mid 10S \mid 11S$$

Problem 8.4: The grammar we were asked to determine is

$$S \to 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$$

P.9 ➡ *Solution*

Language of Problem 8.1: A PDA that never enters an accept state can be used to model a context-free grammar that generates a language given solely by the empty set.

Language of Problem 8.2: This language is regular. The PDA reads the input and keeps track of the first and last symbol in its finite control. If they are the same, the machine accepts; otherwise, it rejects.

Language of Problem 8.3: This language is regular. The PDA reads the input and keeps track of the length (modulo-2) using its finite control. If the length is 1 (mod 2), it accepts; otherwise, it rejects.

Language of Problem 8.4: The PDA reads the input and pushes each symbol onto its stack. At some point it nondeterministically guesses when it has reached the middle. It also nondeterministically guesses whether the string has odd length or even length. If it guesses even, it pushes the current symbol it's reading onto the stack. If it guesses the string has odd length, it goes to the next input symbol without changing the stack. Then it reads the rest of the input, and it compares each symbol it reads to the symbol on the top of the stack. If they are the same, it pops the stack, and continues reading. If they are different, it rejects. If the stack is empty when it finishes reading the input, it accepts. If the stack is empty before it reaches the end of the input, or nonempty when the input is finished, it rejects.

P.10 ➡ *Solution*

Problem 10.1: The complement of the language $\{a^n b^n \mid n \geq 0\}$ can be described by the following grammar.

$$\begin{aligned} S &\to XbXaX \mid T \mid U \\ T &\to aTb \mid Tb \mid b \\ U &\to aUb \mid aU \mid a \\ X &\to aX \mid bX \mid \varepsilon \end{aligned}$$

Problem 10.2: The language in question can be described as follows.

$$\begin{aligned} S &\to M\#P\#M \mid P\#M \mid M\#P \mid P \\ P &\to aPa \mid bPb \mid a \mid b \mid \varepsilon \mid \# \mid \#M\# \\ M &\to aM \mid bM \mid \#M \mid \varepsilon \end{aligned}$$

Note that we need to allow for the case when $i = j$, that is, some x_i is a palindrome. Also, ε is in the language since it is a palindrome.

P.11 ➡ *Solution*

Problem 11.1: A context-free grammar that generates A can be described as $G = (V, \Sigma, R, S)$, where nonterminal symbol set $V = \{S, E_{ab}, E_{bc}, C, A\}$ and terminal symbol set $\Sigma = \{a, b, c\}$, giving

$$\begin{aligned} S &\to E_{ab}C \mid AE_{bc} \\ E_{ab} &\to aE_{ab}b \mid \varepsilon \\ E_{bc} &\to bE_{bc}c \mid \varepsilon \\ C &\to Cc \mid \varepsilon \\ A &\to Aa \mid \varepsilon \end{aligned}$$

Initially substituting $E_{ab}C$ for S generates any string with an equal number of a's and b's followed by any number of c's. Initially substituting E_{bc} for S generates any string with an equal number of b's and c's prepended by any number of a's. The grammar is ambiguous. Consider the string ε. On the one hand, it can be derived by choosing $E_{ab}C$ with each of E_{ab} and C yielding ε. On the other hand, ε can be derived by choosing AE_{bc} with each of A and E_{bc} yielding ε. In general, any string $a^i b^j c^k$ with $i = j = k$ can be derived ambiguously with this grammar.

Problem 11.2: The automaton is described as follows.

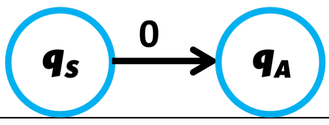
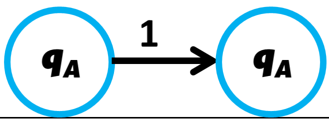
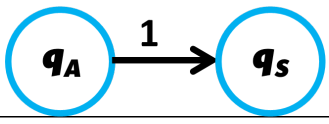
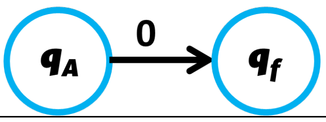
1. Nondeterministically branch to either stage 2 or stage 6.
2. Read and push a's.
3. Read b's, while popping a's.
4. If b's finish then stack is empty, skip c's on input and *accept*.
5. Skip a's on input.
6. Read and push b's.
7. Read c's, while popping b's.
8. If c's finish when stack is empty, *accept*.

P.12 ➡ *Solution*

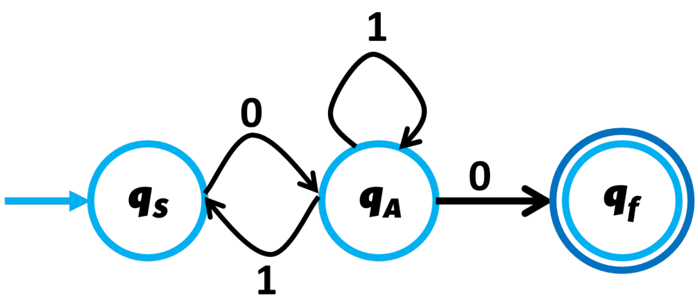
The transition system to be determined can be described as

$$M = (\{q_S, q_A, q_f\}, \{0,1\}, \delta, q_S, \{q_f\})$$

where q_S and q_A correspond to S and A, respectively. q_f is the final state, used for production $A \to 0$. Using the grammar rules, we have the following transitions:

	Production $S \to 0A$
	Production $A \to 1A$
	Production $A \to 1S$
	Production $S \to 0$

These transitions can be combined to yield the following automaton.



P.13 ➡ *Solution*

The system at hand can be described by a grammar $G = (V, \Sigma, R, S)$ with nonterminal symbol set $V = \{S, M, N, P\}$, terminal symbol set $\Sigma = \{0,1\}$, and production rules

$$\begin{gathered} S \to 0M \mid 1N \\ M \to 0M \mid 1P \\ N \to 0P \mid 1N \\ P \to 0P \mid 1P \mid \varepsilon \end{gathered}$$

P.14 ➡ *Solution*

Problem 14.1: The language of automaton M works as follows. First, at initial state q, the automaton pushes the input symbols a and b to the stack until it reads a tape symbol c. After reading c, it moves to state p. In state p, M compares each symbol on the tape with the top symbol of the stack. If they are all equal, then the machine accepts the input. Let $w \in \{a, b\}^*$ be the prefix of the input before the first occurrence of symbol c, and x be the suffix of the input after the first occurrence of c. We note that, as the PDA switches to state p, the string w is stored in the stack in the reverse order (the first symbol of w is at the bottom of the stack, and the last symbol is at the top). So, when it compares the suffix x with the stack symbols, it compares the first symbol of x with the last symbol of w, and so on. Therefore, it only accepts if $x = w^R$. That is,

$$\underline{L(M) = \{wcw^R \mid w \in \{a,b\}^*\}}$$

Problem 14.2: Put simply, the PDA works as follows:
1. Push the first symbol of the input string, whatever it is.
2. If the input symbol and top element of the pushdown system are the same, then push the input symbol to the PDS.
3. If the input symbol and top element of the pushdown system are different (i.e., input is a and top element of PDS is b, or input is b and top element of the PDS is a) then pop the top element of the PDS.

To design the PDA, observe the following from the transition function rules. The first input symbol of the language is pushed onto the empty stack (see transitions T_1 and T_2). As the second symbol of the input is read, the PDA performs a push operation if the input symbol and pop element of the pushdown stack are the same (see transition rules T_3 and T_4), otherwise the PDA performs a pop operation (see rules T_5 and T_6). Accordingly, a single a pops a single b, and a single b pops a single a from the pushdown store. In this manner, a n number of a's pops a n number of b's and vice versa. It follows that if the language holds an equal number of a's and b's, the stack becomes empty after reading the complete input. This shows the acceptability of the required language by null store.

A final note: The PDA addressed in this problem also accepts the language $L' = \{a^n b^n \mid n \geq 0\}$, because it contains an equal number of a's and b's. Language L' can be described by a set of transition rules similar to the one provided above (although T_2, T_4 and T_5 ultimately become useless when describing L').

Problem 14.3: The approach to describe this automaton and finding the transition function rules is similar to the one adopted in Problem 14.2. The PDA reads the first input symbol and pushes it onto the stack. The PDA reads the whole input and carries out a push operation if the input symbol and the entry at the top of the stack are the same; otherwise, the PDA performs a pop operation. This way, the stack does not become empty after reading the full input. There is at least one a in the system stack, indicating that the number of a's is greater than the number of b's.

The pushdown automaton stays in state q_0 until it decides to quit, which it does by checking that the stack has at least on a on it. In state q_0 the PDA is able either to quit or continue reading, so the constructed PDA is nondeterministic. The following rules are used to describe transition function δ,

T_1: $\delta(q_0, a, Z_0) = (q_0, aZ_9)$	If string starts with an a.
T_2: $\delta(q_0, b, Z_0) = (q_0, bZ_9)$	If string starts with an b.
T_3: $\delta(q_0, a, a) = (q_0, aa)$	
T_4: $\delta(q_0, b, b) = (q_0, bb)$	
T_5: $\delta(q_0, a, b) = (q_0, \varepsilon)$	
T_6: $\delta(q_0, b, a) = (q_0, \varepsilon)$	
T_7: $\delta(q_0, \varepsilon, a) = (q_1, a)$	Case when $n_a(x) > n_b(x)$

The deterministic PDA equivalent to the system described above will retain states q_0 and q_1, but it will enter q_1 whenever it reads an a and the stack is empty. This move does not change the status of the pushdown store. If it pushed an a, in order to reflect the fact that there is currently no surplus of a's, there would be no way to determine, at the point in which a was about to be removed from the stack, that the PDA should leave state q_1. The PDA will leave the state q_1 only by reading b when the stack is empty – except for Z_0, that is, by reading b when there is currently one excess a, and that move also leaves the stack unchanged. The deterministic PDA described by the following transition rules has q_1 as final state and there is no move specified from q_1 with stack symbol b or from q_0 with stack symbol a, because neither of these transitions will ever occur.

T_1: $\delta(q_0, a, Z_0) = (q_1, Z_9)$
T_2: $\delta(q_0, b, Z_0) = (q_0, bZ_9)$
T_3: $\delta(q_0, a, b) = (q_0, \varepsilon)$
T_4: $\delta(q_0, b, b) = (q_0, bb)$
T_5: $\delta(q_1, a, Z_0) = (q_1, aZ_0)$
T_6: $\delta(q_1, b, Z_0) = (q_0, Z_0)$
T_7: $\delta(q_1, a, a) = (q_1, aa)$
T_8: $\delta(q_1, b, a) = (q_1, \varepsilon)$

To show that the rules above indeed generate the language we were given, we take a string with more a's than b's – say, *abbabaa* – and apply the pertaining transition rules:

$$(q_0, abbabaa, Z_0) \xrightarrow{T_1} (q_1, bbabaa, Z_0) \xrightarrow{T_6} (q_1, babaa, Z_0)$$

$$\xrightarrow{T_2} (q_0, abaa, bZ_0) \xrightarrow{T_3} (q_0, baa, Z_0) \xrightarrow{T_2} (q_0, aa, bZ_0)$$

$$\xrightarrow{T_3} (q_0, a, Z_0) \xrightarrow{T_1} (q_1, \varepsilon, Z_0) (\text{string accepted by transition } T_1)$$

Problem 14.4: The pushdown automaton that represents language L is somewhat similar to the ones considered in 14.3 and 14.2. Importantly, if the top element of the stack is an a, then the number of a's in the input is greater than the number of b's; similarly, if the top element of the stack is a b, then the number of b's in the input is greater than the number of a's. The transition function δ is described by the following rules,

T_1: $\delta(q_0, a, Z_0) = (q_0, aZ_9)$	
T_2: $\delta(q_0, b, Z_0) = (q, bZ_9)$	
T_3: $\delta(q_0, a, a) = (q_0, aa)$	
T_4: $\delta(q_0, b, b) = (q_0, bb)$	
T_5: $\delta(q_0, a, b) = (q_0, \varepsilon)$	
T_6: $\delta(q_0, b, a) = (q_0, \varepsilon)$	
T_7: $\delta(q_0, \varepsilon, a) = (q_f, a)$	PDA reaches final state q_f with more a's than b's
T_8: $\delta(q_0, \varepsilon, b) = (q_f, b)$	PDA reaches final state q_f with more b's than a's

Problem 14.5: The language at hand can be restated as $L = \{a^n(bbb)^n \mid n \geq 0\}$. The basic principles for designing this PDA are (1) all a's should be pushed onto the stack after scanning, and (2) when b's are scanned, a group of three b's pops one a. The required PDA is described by the tuple

$$M = (\{q_0, q_1, q_2, q_3, q_f\}, \{a,b\}, \{a, Z_0\}, \delta, Z_0, q_0, \{q_f\})$$

where transition function δ is described by the following rules.

T_1: $\delta(q_0, \varepsilon, Z_0) = (q_f, Z_0)$	Terminating transition if $\varepsilon \in L$ (a case for $n = 0$).
T_2: $\delta(q_0, a, Z_0) = (q_0, aZ_0)$	First a is pushed onto stack.
T_3: $\delta(q_0, a, a) = (q_0, aa)$	Additional a's are pushed onto stack.
T_4: $\delta(q_0, b, a) = (q_1, a)$	First b is scanned from a group of three b's.
T_5: $\delta(q_1, b, a) = (q_2, a)$	Second b is scanned from a group of three b's.
T_6: $\delta(q_2, b, a) = (q_3, \varepsilon)$	Third b from group of three b's is scanned; one a is popped.
T_7: $\delta(q_3, b, a) = (q_1, a)$	First b is scanned from next group of bbb.
T_8: $\delta(q_3, \varepsilon, Z_0) = (q_f, \varepsilon)$	The input string is exhausted and accepted in the final state.

The PDA at hand is based on acceptability by final state. To consider acceptability by null (empty) store we need to replace transitions T_1 and T_8 with

$$T_1: \delta(q_0, \varepsilon, Z_0) = (q_0, Z_0)$$

$$T_8: \delta(q_3, \varepsilon, Z_0) = (q_3, Z_0)$$

respectively, yielding an automaton $M = (\{q_0, q_1, q_2, q_3\}, \{a,b\}, \{a, Z_0\}, \delta, q_0, Z_0, \{\emptyset\})$.

Problem 14.6: The solution automaton can be divided into two parts. In one part, the PDA compares a's and b's and arrives at the final state by ensuring that either $i > j$ or $i < j$. In the other part, the PDA compares b's with c's and reaches the final state while ensuring that either $k > j$ or $k < j$. The transition rules for this PDA are shown below.

T_1: $\delta(q_0, a, Z_0) = (q_0, aZ_0)$	The first a is pushed onto stack.
T_2: $\delta(q_0, a, a) = (q_0, aa)$	The remaining a's are pushed onto stack.
T_3: $\delta(q_0, b, a) = (q_1, \varepsilon), (q_2, b)$	First b is scanned.
T_4: $\delta(q_1, b, a) = (q_1, \varepsilon)$	Second-to-last b is scanned.
T_5: $\delta(q_1, b, Z_0) = (q_f, Z_0)$	$i < j$, PDA reaches final state q_f.
T_6: $\delta(q_1, c, a) = (q_f, a)$	$i > j$, PDA reaches final state q_f.
T_7: $\delta(q_2, b, b) = (q_2, bb)$	
T_8: $\delta(q_2, c, b) = (q_2, \varepsilon)$	
T_9: $\delta(q_2, c, a) = (q_f, a)$	$j < k$, PDA reaches final state q_f.
T_{10}: $\delta(q_2, \varepsilon, b) = (q_f, b)$	$j > k$, PDA reaches final state q_f.

P.15 ➡ *Solution*

To obtain the Chomsky normal form (CMF) of a context-free diagram, we proceed as follows. Suppose the CFG we begin with has general form $G = (V, \Sigma, P, S)$.

Step 1: Simplification. Simplify the grammar by performing null and unit production eliminations. These are not covered herein but can be found in Chapter 6 of Kumar (2010). In the simplest possible terms, a context-free grammar with no null productions has no relations of the form $A \to \varepsilon$, while a CFG with no unit productions has no relations of the form $A \to B$, where A and B are nonterminals. Let the grammar obtained after simplification be $G' = (V', \Sigma', P', S)$.

Step 2: Elimination of terminals from right-hand side. Let grammar $G'' = (V'', \Sigma', P'', S)$ be the grammar obtained in this step. P'' and V'' are constructed as follows.

2.1 → All the production rules in P' that have the form $A \to b$ or $B \to BD$ are included in P'', and all the variables in V' are included in V''.

2.2→ For productions of the form $X \to A_1A_2 \ldots A_n$ with one or more terminals on the right-hand side, then for every terminal A_i associated with some value a_i we create a new variable C_{a_i} associated with a new rule $C_{a_i} \to a_i$. Then, we add variable C_{a_i} to V'' and the new production rule to P''.

Every terminal on the right-hand side of the production $X \to A_1A_2 \ldots A_n$ is replaced by the corresponding new variable produced in 2.2, and the variables on the right-hand side are retained. The resulting productions are added to set P'', which gives the updated grammar G''.

Step 3. Restricting the number of variables on right-hand side. Every production in P'' must contain either a single terminal (or ε in the case $S \to \varepsilon$) or two or more variables on the right-hand side. We define grammar $G''' = (V''', \Sigma', P''', S)$ as follows:

3.1 → All production rules P'' that obey the requirement above are included in P'''; all variables in V'' are included in V'''.

3.2 → If there are productions of the form $A \to A_1A_2 \ldots A_n$, where $n \geq 3$, we introduce production rules $A \to A_1C_1$, $C_1 \to A_2C_2$, $C_2 \to A_3C_2$, $C_3 \to C_3A_4$, ... $C_{n-2} \to A_{n-2}A_n$ and new variables $C_1, C_2, C_3, \ldots, C_{n-2}$. These productions are added to P''' and the new variables C_i produced are added to V'''.

Once step 3 is completed, the resulting grammar G''' will be in Chomsky normal form.

Notice that the grammar we were given has neither null productions nor unit productions, so we can skip step 1 altogether. At this point, we have grammar $G' = (V', \Sigma', P', S)$, where

$$V' = \{S, A, B, C\}$$

$$\Sigma' = \{a, b, c\}$$

$$P' = \{S \to aAC, A \to aB \mid bAB, B \to b, C \to c\}$$

$$S \text{ is the start symbol, and } S \in V'$$

We proceed to step 2, in which we have to eliminate terminals from the right-hand side of production rules. Rules $B \to b$ and $C \to c$ are already in desired form, but $S \to aBC$ and $A \to aB \mid bAB$ must be reworked. We introduce a new variable $C_a \to a$ and restate $S \to aBC$ as $S \to C_aBC$. Also, $A \to aB$ is updated as $A \to C_aB$. A second variable $C_b \to b$ is introduced, and rule $A \to bAB$ becomes $A \to C_bAB$. This concludes step 2. At this point, we have $G'' = (V'', \Sigma', P'', S)$, where

$$V'' = \{S, A, B, C, C_a, C_b\}$$

$$P'' = \left\{\begin{matrix} S \to C_aAC,\ C_a \to a,\ A \to C_aB | C_bAB,\ C_b \to b, \\ B \to b,\ C \to c \end{matrix}\right\}$$

We move on to step 3, where we have to restrict the number of variables on the right-hand side of production rules in P_i. Note that productions $C_a \to a$, $A \to C_aB$, $C_b \to b$, $B \to b$, and $C \to c$ are all in required form, but $S \to C_aAC$ and $A \to C_bAB$ are not. To deal with these cases, we rewrite the former as $S \to C_aC_1$, where C_1 is a new variable such that $C_1 \to AC$, and the latter as $A \to C_bC_2$, where $C_2 \to AB$. Variables C_1 and C_2 are added to V''', and the production rules listed in this paragraph are added to *P*'''. At this point, we have grammar *G*''' = (*V*''', Σ', *P*''', *S*), where

$$V''' = \{S, A, B, C, C_a, C_b, C_1, C_2\}$$

$$P''' = \left\{ \begin{matrix} S \to C_aC_1,\ C_a \to a,\ A \to C_aB|C_bC_2,\ C_b \to b, \\ B \to b,\ C \to c,\ C_1 \to AC,\ C_2 \to AB \end{matrix} \right\}$$

Grammar *G*''' is in Chomsky normal form, as intended.

REFERENCES

- HOPCROFT, J.E., MOTWANI, R. and ULLMAN, J.D. (2001). *Introduction to Automata Theory*, *Languages, and Computation*. 2nd edition. Upper Saddle River: Addison-Wesley.
- KUMAR, R. (2010). *Theory of Automata*, *Languages and Computation*. New Delhi: Tata-McGraw-Hill.
- LINZ, P. (2017). *An Introduction to Formal Languages and Automata*. 6th edition. Burlington: Jones and Bartlett Learning.
- SIPSER, M. (2013). *Introduction to the Theory of Computation*. 3rd edition. Boston: Cengage Learning.