CS 162 **Fall 2015**

Homework 4 Problems

October 21, 2015 *Timothy Johnson*

1. Let $L = \{0^n 1^{2n} | n > 0\}$ Give a CFG for L.

   The following CFG will produce L.

   $$S \rightarrow 0S11 \mid 011 \mid \epsilon$$

2. Show that every regular language L is also context free. Hint: use a proof by induction on the number of operators in a regular expression for L.

   Base case: we will start with single characters, with no operations. To recognize a single character a, we can just have a CFG with a single transition: $S \rightarrow a$.

   Inductive step: Now suppose that for any regular expression with fewer than k operators we can construct a CFG that produces the same language. Then take a language L be represented by the regular expression R, such that R has k operators.

   We have three possible operations.

   - Union: If $R = R_1 + R_2$, then assume that R_1 is produced by a CFG with start symbol S_1, and that R_2 is produced by a CFG with start symbol S_2. We create a new start state S with the production rule $S \rightarrow S_1 \mid S_2$. If the first production we use is $S \rightarrow S_1$, then we will produce exactly the strings matched by R_1, by our inductive hypothesis. If the first production we use is $S \rightarrow S_2$, then we will produce exactly the strings matched by R_2.
   - Concatenation: If $R = R_1 R_2$, then we create a new CFG with start state S and the production rule $S \rightarrow S_1 S_2$. By our inductive hypothesis, S_1 produces exactly the strings matched by R_1, and S_2 produces exactly the strings matched by R_2. So S will produce exactly the strings in $R_1 R_2$.
   - Kleene star: If $R = R_1^*$, then we create a new CFG with start state $S \rightarrow S_1 S_1 \mid \epsilon$. Then we can prove by induction (details omitted) that a string with any number of copies of strings matched by R_1 can be generated.

   Therefore, since every regular expression has an equivalent CFG, every regular language is context free.

3. Exercise 5.1.2 on page 182 of Hopcroft et al.
The following grammar generates the language of the regular expression $\mathbf{0}^*\mathbf{1}(\mathbf{0}+\mathbf{1})^*$.

$$\begin{aligned} S &\rightarrow A1B \\ A &\rightarrow 0A \mid \epsilon \\ B &\rightarrow 0B \mid 1B \mid \epsilon \end{aligned}$$

Give leftmost and rightmost derivations of the following strings:

(a) 00101

Leftmost	Rightmost
$S \rightarrow A1B$	$S \rightarrow A1B$
$\rightarrow 0A1B$	$\rightarrow A10B$
$\rightarrow 00A1B$	$\rightarrow A101B$
$\rightarrow 001B$	$\rightarrow A101$
$\rightarrow 0010B$	$\rightarrow 0A101$
$\rightarrow 00101B$	$\rightarrow 00A101$
$\rightarrow 00101$	$\rightarrow 00101$

(b) 1001

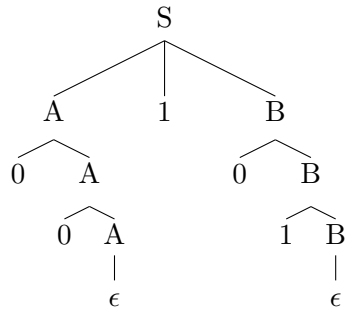
Leftmost	Rightmost
$S \rightarrow A1B$	$S \rightarrow A1B$
$\rightarrow 1B$	$\rightarrow A10B$
$\rightarrow 10B$	$\rightarrow A100B$
$\rightarrow 100B$	$\rightarrow A1001B$
$\rightarrow 1001B$	$\rightarrow A1001$
$\rightarrow 1001$	$\rightarrow 1001$

(c) 00011

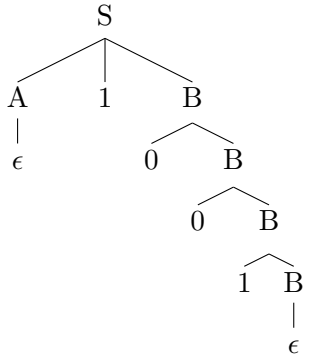
Leftmost	Rightmost
$S \rightarrow A1B$	$S \rightarrow A1B$
$\rightarrow 0A1B$	$\rightarrow A11B$
$\rightarrow 00A1B$	$\rightarrow A11$
$\rightarrow 000A1B$	$\rightarrow 0A11$
$\rightarrow 0001B$	$\rightarrow 00A11$
$\rightarrow 00011B$	$\rightarrow 000A11$
$\rightarrow 00011$	$\rightarrow 00011$

4. Exercise 5.2.1 on page 193 of Hopcroft et al.
For the grammar and each of the strings in Exercise 5.1.2, give parse trees.

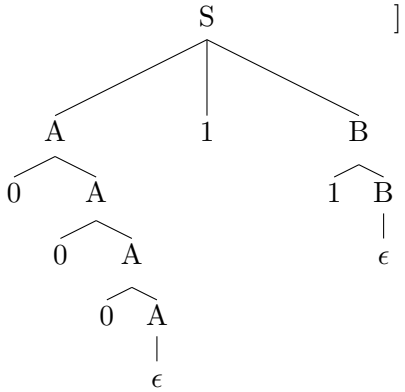
(a) 00101



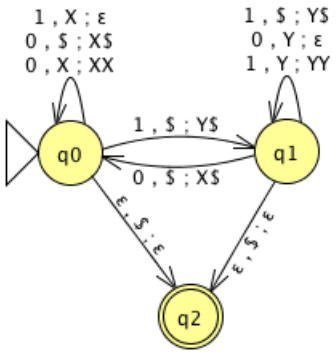
(b) 1001



(c) 00011



5. Let L be the language of all strings w of 0's and 1's such that w has an equal number of 0's and 1's (in any order). Give a PDA for L. Include comments in your answer describing your PDA in English as well as using a transition function.



In state q_0 we have seen at least as many 0's as 1's. We keep a number of X's on our stack that is equal to the number of 0's minus the number of 1's. For each 0 we push a new X, and for each 1 we pop a Y.

In state q_1 we have seen at least as many 1's as 0's. We keep a number of Y's on our stack that is equal to the number of 1's minus the number of 0's. For each 1 we push a new Y, and for each 0 we pop a Y.

At any point when our stack is empty, we can choose to transition to the final state q_2. If we have finished processing our string, we accept, because we must have seen an equal number of 0's and 1's. But if there is still more input, that branch of our computation will crash, and we will continue moving between q_0 and q_1.