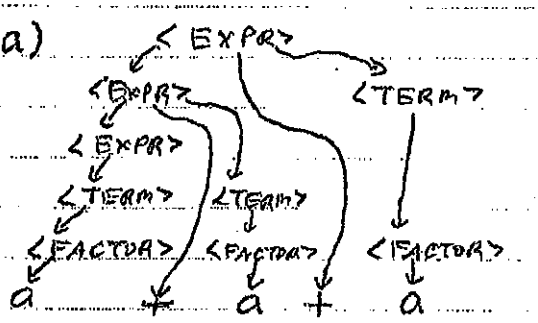
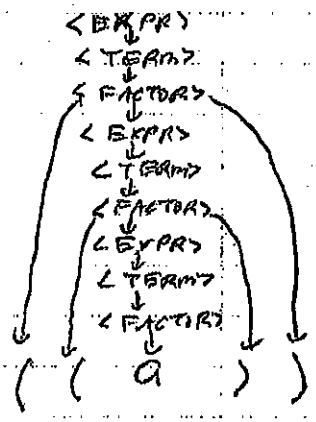
COT 4210 DAILY PROOF SOLUTIONS SECTION 2:1

1) a)



b)



2) a) $S \to 0S0 \mid 1S1 \mid 1 \mid 0 \mid \varepsilon$

All palindromes are built by padding any 0 or 1 length string with the same character at the front and end repeatedly. The first two rules enforce the latter idea and the last three are the "base case" strings.

b) $S \to \emptyset S \mid 1A$
$A \to \emptyset A \mid 1B$
$B \to \emptyset B \mid 1C$
$C \to \emptyset C \mid 1C \mid \varepsilon$

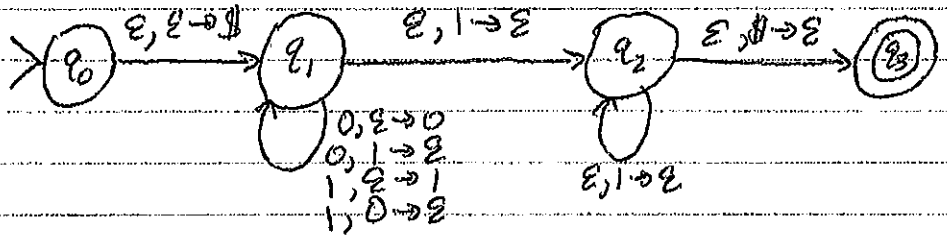
Follows the DFA → CFG construction. S = zero 1's, A = one 1, B = two 1's, C = three or more 1s are what each VARIABLE/STATE stands for.

c) $S \to \emptyset S \emptyset \mid \emptyset S 1 \mid 1 S \emptyset \mid 1 S 1 \mid 0$

The only "base case" is 0. We successively add a single character to the left and right of this middle 0 to create a string in the language.

d) $S \to A \mid A$
$A \to A1A \mid A1A0A \mid A0A1A \mid \varepsilon$

We force at least one 1 by going from S to A. Any time we add a 0, we also add a 1, not restricting order.

3) 

$Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \$\}$, δ is above, q_0 is the start state, $F = \{q_3\}$.

q_0 marks starting the string processing by requiring $ at the bottom of the stack.

q_1 is the state where all string processing occurs. We keep track of a symbol by either pushing it on the stack or popping its "opposite" off the stack.

q_2 forces there to be at least one 1 on the stack after processing all characters. We ALSO ensure that NO zeros are on the stack by providing no transitions from q_2 that pop a 0.

We can only get to q_3 after getting to q_2 with no zeros on the stack, after ALL extra 1s have been popped. We can finally accept here. q_2 can't be an accept state because we can get there by having a bunch 0s at the bottom of the stack with one 1 at the top.

4) ⓐ Add a new start symbol

$S \to A$

$A \to BAB \mid B \mid \varepsilon$

$B \to 00 \mid \varepsilon$

ⓑ Remove ε rules (except $S \to \varepsilon$)

$S \to A \mid \varepsilon$

$A \to BAB \mid BA \mid AB \mid B \mid BB$

$B \to 00$

ⓒ Remove unit rules

$S \to BAB \mid BA \mid AB \mid BB \mid 00 \mid \varepsilon$

$A \to BAB \mid BA \mid AB \mid BB \mid 00$

$B \to 00$

ⓓ Convert long rules, terminals

$S \to \varepsilon \mid BD \mid BA \mid AB \mid BB \mid CC$

$A \to BD \mid BA \mid AB \mid BB \mid CC$

$B \to CC$

$C \to 0$

$D \to AB$

5) Let G_1 be a context-free grammar (V_1, Σ, R_1, S_1), and G_2 be a context-free grammar (V_2, Σ, R_2, S_2). We will prove that (a) $L(G_1) \cup L(G_2)$, (b) $L(G_1) \circ L(G_2)$ and (c) $(L_*(G_1))^*$ are all languages that are context-free.

(a) We will create a CFG G such that $L(G) = L(G_1) \cup L(G_2)$. Let $G = (V, \Sigma, R, S)$ with

$V = V_1 \cup V_2 \cup \{S\}$

$R = R_1 \cup R_2 \cup \{S \to S_1 \mid S_2\}$

Now we show $L(G) = L(G_1) \cup L(G_2)$. Consider ~~and~~ any string $w \in L(G)$. The first derivation must be either $S \to S_1$ or $S \to S_2$. If w is produced on the first "path" then $w \in L(G_1)$, otherwise is w is produced on the second path then $w \in L(G_2)$. By defn of union it follows that $w \in L(G_1) \cup L(G_2)$. Now consider any string $w \in L(G_1) \cup L(G_2)$. Either $w \in L(G_1)$ or $w \in L(G_2)$. In the first case, we prove $w \in L(G)$ by taking the derivation $S \to S_1$ and replicating the steps of deriving w in G_1. In the second case we also show $w \in L(G)$ by taking the derivation $S \to S_2$ and replicating the steps of deriving w in G_2. Thus we've shown $w \in L(G)$. It follows that $L(G) = L(G_1) \cup L(G_2)$, proving that context free languages are closed under the union operation

5) (b) We will create a CFG G such that $L(G) = L(G_1) \cdot L(G_2)$.

Let $G = (V, \Sigma, R, S)$ with

$V = V_1 \cup V_2 \cup \{S\}$

$R = R_1 \cup R_2 \cup \{S \to S_1 S_2\}$

Now we show $L(G) = L(G_1) \cdot L(G_2)$. Consider any string $w \in L(G)$. Since the first production must be $S \to S_1 S_2$, there exist strings u, v such that u is derived by S_1 and v in derived by S_2 with $w = uv$. It follows that $u \in L(G_1)$ and $v \in L(G_2)$. Thus, $w \in L(G_1) \cdot L(G_2)$. Now consider any string $w \in L(G_1) \cdot L(G_2)$. There exist u, v with $u \in L(G_1)$ and $v \in L(G_2)$ with $w = uv$. In G, do the production $S \to S_1 S_2$ and then replicate the productions for S_1 to produce u and in S_2 to produce v. In this manner G produces uv. Thus $uv = w \in L(G)$, as desired. It follows that $L(G) = L(G_1)L(G_2)$ and context free languages are closed under concatenation.

(c) We will create a CFG G such that $L(G) = (L(G_1))^*$.

Let $G = (V, \Sigma, R, S)$ with

$V = V_1 \cup \{S\}$

$R = R_1 \cup \{S \to \varepsilon, S \to S_1 S\}$

Now we will show $L(G) = (L(G_1))^*$.

Consider any string $w \in L(G)$. w must have been produced by some number of applications of $S \to S_1 S$ and one application of $S \to \varepsilon$. Thus w must be produced by some number of replications of S_1. By definition of $*$ and S_1, it follows that $w \in (L(G_1))^*$. Now, consider any string $w \in (L(G_1))^*$. There exist $u_1, u_2, \ldots, u_k$ such that $w = u_1 u_2 \cdots u_k$ and $u_i \in L(G_1)$ for $1 \le i \le k$. We produce w in G by repeating the production $S \to S_1 S$ k times and using $S \to \varepsilon$. Finally, we use the i^{th} variable S_1 from the left to produce u_i; proving $w \in L(G)$. It follows that $L(G) = ((L(G_1))^*$ and context free languages are closed under $*$.

6) G = (V, Σ, R, S) with

V = {S, T, U, V, W}
Σ = {a, b, c}
Start symbol = S

R: $S \rightarrow aT$
$T \rightarrow aT \mid U$
$U \rightarrow VW$
$V \rightarrow aVb \mid \varepsilon$
$W \rightarrow cW \mid \varepsilon$

The first production guarantees that there is at least one a unmatched by a b. T allows us to have any number more a's without a matched b. U forces us to have a string with two separate components, the first component, V, which allows an addition of matched a's and b's in the appropriate place in the string while the second component, W, allows us to add any number of c's. Intuitively, to derive the string $a^ib^jc^k$, we use the production for S, followed by the production for T i-j-1 times, followed by using the production for U once, followed by using the production for V j times, and then the production for W k times.

This grammar is unambiguous due to its directed nature. Productions such as S and U must occur exactly once, while all productions of T must occur in between the productions of S and U. This forces a structure for the parse tree of the derivation, since there is only one rule for T that doesn't lead to U. Similarly, due to the fact that both V and W have only one rule that doesn't "terminate" the derivation, both of these create unambiguous portions of the total parse tree, making the entire parse tree for any given string, unambiguous.

7) Our PDA will have four states: q_{start}, q_a, q_b, and q_c with the start state q_{start}, the alphabet {a,b,c}, the stack alphabet of {$,a}, the accept state q_c, and the following transition function:

$\delta(q_{start}, a, \varepsilon) = \{(q_a, \$)\}$, push a $ on the stack after reading the first a.
$\delta(q_a, a, \varepsilon) = \{(q_a, a)\}$, add an a to the stack for each subsequent a.
$\delta(q_a, \varepsilon, \varepsilon) = \{(q_b, \varepsilon)\}$, non-deterministically move to q_b to get ready to read in b's.
$\delta(q_b, b, a) = \{(q_b, \varepsilon)\}$, must match each b with an "extra" a on the stack.
$\delta(q_b, \varepsilon, \varepsilon) = \{(q_c, \varepsilon)\}$, non-deterministically move to q_c to get ready to read in c's.
$\delta(q_c, c, \varepsilon) = \{(q_c, \varepsilon)\}$, only stay in q_c with any additional c's.